Scan-Based Movement-Assisted Sensor Deployment Methods in Wireless Sensor Networks

Shuhui Yang, Minglu Li, and Jie Wu

Abstract—The efficiency of sensor networks 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, a good sensor deployment is still necessary to balance the workload of sensors. In a sensor network with locomotion facilities, sensors can move around to self-deploy. The movement-assisted sensor deployment deals with moving sensors from an initial unbalanced state to a balanced state. Therefore, various optimization problems can be defined to minimize different parameters, including total moving distance, total number of moves, communication/computation cost, and convergence rate. In this paper, we first propose a Hungarian-algorithm-based optimal solution, which is centralized. Then, a localized Scan-based Movement-Assisted sensoR deploymenT method (SMART) and several variations of it that use scan and dimension exchange to achieve a balanced state are proposed. An extended SMART is developed to address a unique problem called *communication holes* in sensor networks. Extensive simulations have been done to verify the effectiveness of the proposed scheme.

Index Terms—Dimension exchange, Hungarian method, load balance, movement-assisted, scan, sensor deployment, wireless sensor networks.

1 Introduction

Wireless sensor networks (WSNs) [1], [2] combine processing, sensing, and communications to form a distributed system that is capable of self-organizing, self-regulating, and self-repairing. The application of WSNs ranges from environmental monitoring to surveillance to coordinated target detection. 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 necessary to balance the workload of sensors. Mobile sensors [3] can be exploited to provide a redistribution.

After an initial random deployment of sensors in the field, *movement-assisted sensor deployment* [4] can be applied, which uses a potential-field-based approach to move existing sensors by treating sensors as virtual particles subject to virtual forces. Basically, movement-assisted sensor deployment deals with moving sensors from an initial unbalanced state to a balanced state. Therefore, various optimization problems can be defined to minimize different parameters, including total moving distance, total number of moves, communication/computation cost, and convergence rate.

More recently, some extended virtual force methods, such as those in [5] and [6], which are based on disk packing theory [7] and the virtual force field concept from robotics [8], are proposed. These methods simulate the attractive and repulsive forces between particles. Sensors in a relatively dense region will explode slowly according to each other's repulsive force and head toward a sparse region. In this way, the whole monitoring area can achieve an even distribution of sensors. However, these methods may have long deployment times since sensors move independently, and they may even fail if all the sensors can achieve force balance but not load balance.

We assume that sensors are deployed randomly into the square monitoring area without consideration of any physical obstacles. Then, if we partition the monitoring area into many small regions and use the number of sensors in a region as its load, the sensor deployment problem can be viewed as a load balance problem in traditional parallel processing, where each region corresponds to a processor and the number of sensors in a region corresponds to the load. The sensor deployment resembles the traditional load balance issue in parallel processing, with several key differences:

- *Different objectives*. In traditional load balancing, the total moving distance, rather than the number of moves, is important, whereas in sensor networks, the number of moves is also important because of a relatively heavy energy consumption to start or stop a move.

- S. Yang and J. Wu are with the Department of Computer Science and Engineering, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431. E-mail: {syang1, jie}@cse.fau.edu.
- M. Li is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, 1954 Huashan Road, Shanghai, PR China. E-mail: li-ml@cs.stju.edu.cn.

Manuscript received 1 June 2006; revised 28 Sept. 2006; accepted 2 Oct. 2006; published online 9 Jan. 2007.
Recommended for acceptance by S. Olariu.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0140-0606. Digital Object Identifier no. 10.1109/TPDS.2007.1048.

- *Different technical issues*. One unique issue in sensor networks is the communication hole (or simply "hole") problem, where some regions of the network have no deployed sensors. Since there is no centralized control, the network can be partitioned. Therefore, the network needs to be connected prior to load balancing.

In this paper, we first provide an optimal solution in 2D meshes. This solution is based on the classic Hungarian method but requires global information without considering sensor network connectivity. We then propose a method using a 2D scan called Scan-based Movement-Assisted sensoR deploymenT method (SMART). A typical scan operation [9] involves a binary operator $\oplus$ and an ordered set $[w_0, w_1, \ldots, w_{n-1}]$, where each w_i represents the number of sensors in a region and returns the ordered set

$$[w_0, (w_0 \oplus w_1), \ldots, (w_0 \oplus w_1 \oplus, \ldots, \oplus w_{n-1})].$$

In this paper, we consider only integer addition and Boolean AND operations for the scan. By using integer addition, the scan operation will return the partial and total sums of the number of sensors. Since each region position and n are known, the 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.

In SMART, a given rectangular sensor field is first partitioned into a 2D mesh through clustering. Each cluster corresponds to a square region and has a clusterhead that is in charge of bookkeeping and communication with adjacent clusterheads. A hybrid approach is used for load balancing, where the 2D mesh is partitioned into one-dimensional (1D) arrays by row and by column. Two scans are used in sequence: one for all rows and another 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 clusters. Load is shifted from overloaded clusters to underloaded clusters in an optimal way to achieve a balanced state. By optimal, we mean the minimum number of moves and minimum total moving distance. By a balanced state, we refer to a state with the maximum cluster size (the number of sensors in a cluster) and the minimum cluster size being different by at most 1.

The *communication hole* problem in a 2D mesh corresponds to a cluster with a cluster size of 0. Clearly, the scan approach cannot be used in a row or column with holes, since clusterheads separated by one or more holes cannot communicate with each other to perform a scan operation. In extreme cases, the 2D mesh may be disconnected as shown in Fig. 1, where the number in each circle corresponds to the cluster size, and sensors in each cluster can communicate with sensors in adjacent clusters as well as sensors in the same cluster. In Fig. 1, the network is partitioned into two components. Our solution to the hole issue is based on planting a "seed" from a nonempty cluster to an adjacent empty cluster. Various solutions are proposed in such a way that this seed-planting process (also called preprocessing) can be easily integrated with the normal 2D scan process to achieve a good balance of various objectives. The network can use newly developed

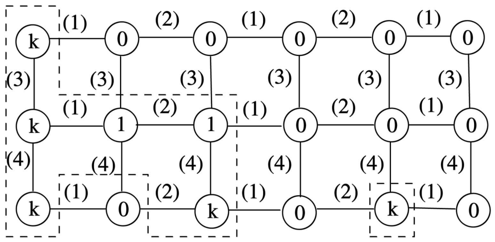

Fig. 1. A sample clustered sensor network that corresponds to a 2D mesh.

location services [10], [11] to estimate the locations of sensors. Thus, no GPS service is required at each sensor, and the corresponding overhead is avoided. For example, locations of sensors can be determined by using sensors themselves as landmarks [12].

The contributions of this paper are listed as follows:

1. We develop an optimal load balance solution based on the classic Hungarian method that achieves the minimum total moving distance and use it as a baseline to check the performance of other approaches.
2. We systematically discuss the similarities and differences between the traditional load balancing in parallel processing and the movement-assisted sensor deployment in sensor networks.
3. We propose a new hybrid approach called SMART, together with several variations, that combines some desirable features of both local and global approaches while overcoming their drawbacks.
4. We identify a unique technical problem called the communication hole and provide solutions to it.
5. We systematically study different trade-offs among various contradictory goals.
6. We conduct extensive simulations and compare the results with several existing local movement-assisted sensor deployment methods.

2 Preliminaries and Related Works

2.1 Load Balance in Multiprocessor Systems

Extensive work on load balancing has been done in the parallel processing community. In general, load balance algorithms can be classified as local (such as iterative nearest neighbor exchanging [13], [14]) and global (such as direct mapping [15], [16]). The global approach relies on global information that is usually not scalable. Local algorithms can be either deterministic or stochastic. Diffusion (DIFF) and dimension exchange (EXCH) are two widely used local deterministic methods. Both algorithms are iterative and are based on a nearest neighbor exchange. Once all nodes complete one iteration, it is called a *sweep*. Although no information on load distribution is needed in local methods, iterative methods incur a significant number of rounds (moves in sensor networks).

In the DIFF method, the balancing procedure is divided into a sequence of synchronous steps. At each step, each node i interacts and exchanges load with all its neighbors $adj(i)$. A DIFF parameter decides the portion of the excess load to be diffused between nodes i and each of its neighbors. Xu and Lau [17] proved that the optimal uniform DIFF parameter that leads to the fastest convergence for 2D meshes is $1/4$.

In the EXCH method, 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. In Fig. 1, all edges are grouped into four dimensions. Edges with label (i) belong to dimension i $(i = 1, 2, 3, 4)$. 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 their load according to an exchange rate. Again, Xu and Lau [17] showed the optimal uniform exchange rate for $2k_1 \times 2k_2$ 2D meshes (where both row and column numbers are even).

2.2 Movement-Assisted Sensor Deployment

The sensor placement issue has been researched recently [18], [19], [20]. A random placement of sensors may not satisfy the deployment requirement due to a hostile deployment environment. Therefore, the movement-assisted sensor deployment method is developed. Most existing movement-assisted 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 the nearest neighbor exchange in load balancing. Sensors are involved in a sequence of computation (for a new position) and movement.

In [6], Zou and Chakrabarty proposed a centralized virtual-force-based mobile sensor deployment algorithm (VFA), which combines the ideas of potential field and disk packing [7]. In VFA, there is a powerful clusterhead that will communicate with all the other sensors, collect sensor position information, and calculate forces and desired position for each sensor. In VFA, the distance between two adjacent nodes when all nodes are evenly distributed is defined as a threshold to distinguish an attractive or a repulsive force between two nodes. The force between two nodes is 0 if their distance is equal to the threshold, attractive if less, and repulsive if greater. The total force on a node is the sum of all the forces given by other sensors together with obstacles and preferential coverage in the area. The clusterhead executes VFA and directs each sensor's movement. VFA has the drawbacks of centralized algorithms, single point of failure, bottleneck of processing, and less scalability.

In [5], Wang et al. developed a novel distributed self-deployment protocol for mobile sensors. They used Voronoi diagrams [21] 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, then 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 [22], they 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 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. In their recent work [23], they designed a virtual movement scheme for the deployment protocol to reduce the moving distance of sensors. To our best knowledge, our work is the first to exploit scan-based movement-assisted solution for sensor redistribution.

Some recent work focus on sensors with limited mobility, which is motivated by the DARPA project called Intelligent Mobile Land Mine Units (IMLM) [24]. In IMLM, the mobility system is based on a hopping mechanism. Chellapan et al. [25] studied a special hopping model in which each sensor can flip (or flop) to a new location only once. In addition, the flip distance is bounded. The deployment problem is then formulated as a minimum-cost, maximum-flow problem.

3 An Optimal Solution

This section starts with an optimal solution for 2D meshes based on the classic Hungarian method. Although, due to its potential drawback of centralization, this optimal solution is not practical, especially when the WSNs are not connected, we can use it as a baseline to examine the performance of other proposed methods.

3.1 Hungarian Method

Let us consider the *edge-weighted matching problem* in a complete bipartite graph $K_{m,m}$ (m nodes on the left side and m on the right), with numbers associated with the 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). The matching finds m edges to connect nodes on the left side to those on the right, and each node has only one edge.

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 the 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 and 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 that it is not. c_{ij} is the weight of edge (v_i, u_j). An optimal solution is

$$\begin{aligned} &\text{Minimize} && \Sigma_{ij} c_{ij} x_{ij} \\ &\text{subject to} && \textstyle\sum_{j=1} x_{ij} = 1 \quad i = 1, \ldots, m \\ &&& \textstyle\sum_{i=1} x_{ij} = 1 \quad j = 1, \ldots, m. \end{aligned}$$

To use the Hungarian method to load balance in WSNs, we need to first convert the 2D mesh to a complete bipartite

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

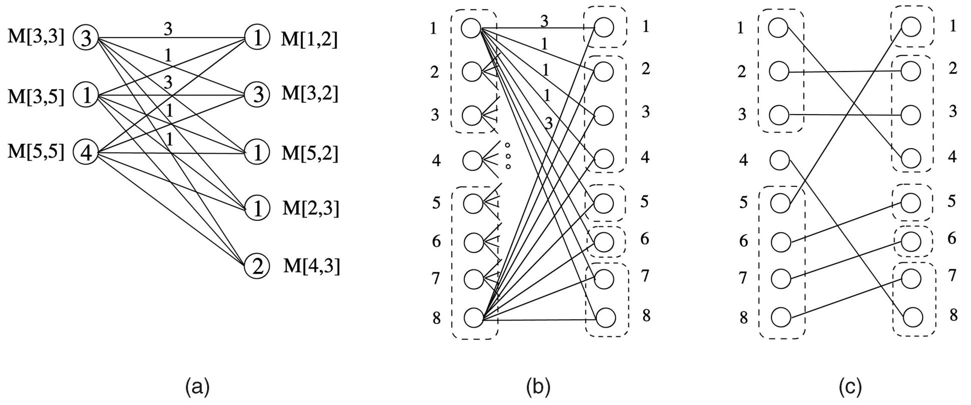

Fig. 2. (a) The node- and edge-weighted bipartite graph of Fig. 3, with "give" grids at the left-hand side and "take" grids at the right-hand side. (b) The edge-weighted complete bipartite graph of (a). (c) The optimal solution.

graph by using the following procedure: 1) Calculate the global average $\bar{v}$ and determine the "give," "take," and "neutral" states of each grid. 2) Construct a node-weighted and edge-weighted bipartite graph, where the "give" and "take" grids appear at the left-hand and right-hand sides of the graph, respectively. The node weight corresponds to the amount of overload and underload, and the edge weight represents the distance between the "give" and "take" grids in a matching pair. 3) Derive an edge-weighted perfect bipartite graph by expanding each node with weight k to k "clone" nodes. The edge weight of the clone nodes will inherit from the original nodes. It is obvious that the total sensor moving distance is minimized. The total number of moves is also minimized, since each sensor, if necessary, only moves once to its destination.

3.2 Examples and Analysis

In Fig. 3, the global average in case is 5. There are three overloaded nodes and five underloaded nodes. $M[3,3]=3$ means overloaded by three units, and $M[1,2]=1$ means underloaded by one 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 Fig. 2a, the node and edge-weighted bipartite graph shows the weights of all edges connecting $M[3,3]$ to underloaded nodes. In Fig. 2b, the edge-weighted complete bipartite graph of Fig. 2a 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 as shown in Fig. 2b, and the optimal result is shown in Fig. 2c. 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 $M[4,3]$.

There are several polynomial implementations for the Hungarian method. Our implementation is based on Munkres' method [26]. Another implementation [27] solves the problem in $O(m^3)$, exploiting the solution to the maximum flow problem. The cost of implementing the Hungarian method for load balance in WSNs is $O(m^3)$, where m is the amount of overloads (underloads) that is bounded by the number of sensors. Usually, the number of sensors is one or two magnitudes higher than the number of grids (n). A base station (BS) is needed to connect to the WSN, serving as the central controller for information collection and algorithm execution. Then, the BS informs all clusterheads about the sensor movement via direct or multihop communication.

4 SMART

4.1 Basic Ideas

Unlike the optimal solution, SMART is a hybrid of the local and global approaches. Its extension (discussed in Section 5) can be used in disconnected WSNs. The sensor network is partitioned into an $n \times n$ 2D mesh of clusters (the method can be easily extended to the general $n \times m$ 2D mesh). Each cluster covers a small square area and is controlled by a clusterhead. The role of each clusterhead can be rotated within the cluster. Each clusterhead, in charge of communicating with adjacent clusters, knows the following information: 1) its cluster position i in the 2D mesh (via GPS) and 2) the number of sensors w_i in the cluster.

Two rounds of balancing are used, one for each dimension: rows first, then columns. As shown in Fig. 3b, after the first round, all rows are balanced. After the second round, all columns are balanced, as is the whole area. Although balancing within a row or column can be done either locally such as iterative nearest neighbor interaction or globally such as direct mapping, SMART relies on an extended scan method.

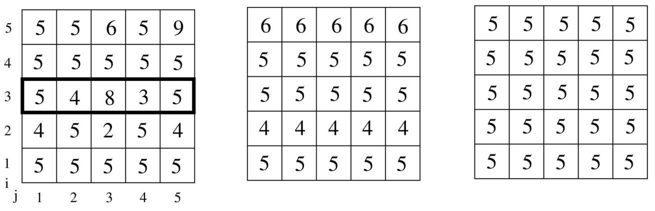

Fig. 3. An ideal case for SMART.

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

4.2 Clustering

Since each sensor node knows its cluster ID i, sensors in the same cluster elect a unique clusterhead based on a predefined priority. Assume that each cluster covers an $x \times x$ square. To ensure that the square is covered whenever there is a sensor in the region, the sensing range r_1 should be set to $\sqrt{2}x$ (the diagonal length of the square). To support the transmission from a nonclusterhead to a clusterhead, the intracluster transmission range should be set to at least $\sqrt{2}x$ (also denoted as r_1). To ensure that the clusterhead can communicate with clusterheads in four adjacent clusters, the intercluster transmission ranges of each clusterhead should be at least the diagonal of the rectangle constructed from two adjacent squares. That is, $r_2 = \sqrt{5}x$. If a sensor does not support two transmission ranges, then r_2 can be used for intracluster communication.

Generally, the role of the clusterhead should rotate among all the nodes in the cluster to achieve a balanced energy consumption and to prolong the life span of each individual node, such as in [28]. Nonclusterheads only need to report their own position and energy to clusterheads by using transmission range r_1, whereas clusterheads will communicate with neighboring clusters, take over the information of sensors in their cluster, and direct the movement of sensors.

4.3 Scan

Consider the 1D array of clusters where the cluster ID is labeled following the sequence in the linear line. Again, denote w_i as the number of sensors in cluster i. Let v_i be the prefix sum of the first i clusters, that is, $v_i = \sum_{j=1}^{i} w_j$. $v_n = \sum_{j=1}^{n} w_j$ 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 that should be rounded to an integer $\lfloor \overline{w} \rfloor$ or $\lceil \overline{w} \rceil$. In a balanced state, $|w_i - w_j| \le 1$ for any two clusters in the network.

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 an increasing order of cluster ID), and that of the second sweep is *negative* (with a decreasing order of cluster ID). 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 cluster. The clusterhead in the last cluster 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 can determine that $\overline{v_i} = i\overline{w}$ (the load of prefix sum in a balanced state) based on $\overline{w}$, which is passed around, and its own cluster position i.

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

TABLE 1
The Scan Process on the Third Row of Fig. 3

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

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

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

The 2D scan process involves a row scan followed by a column scan as shown in Figs. 3b and 3c, respectively. Table 1 shows the details of the row scan on the third row, where i is the column number. Only the cluster at column 3 is in the "give" state, since its load is higher than $\overline{w} = 5$. For column 3, $w_3^{\rightarrow} = 2$ (the load will be assigned to column 4; the actual schedule will be discussed later), and ${}^{\leftarrow}w_3 = 1$ (it will be assigned to column 2). Similarly, a set of conditions can be given for the "take" state: $w_i^{\leftarrow}$ for take-right and ${}^{\rightarrow}w_i$ for take-left. It is clear that

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

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

In the subsequent discussion, we use ${}^{\rightarrow}w_i$ for both the number of take-left units and the take-left state of cluster i. The same convention is used for the other three notations. The distinguishing feature of scan is its simplicity, where each clusterhead in i passes only one package in each sweep: the prefix sum w_i in the first sweep followed by the global average $\overline{w}$ in the second sweep.

4.4 Properties of Scan

An optimal load balance scheduling based on scan should satisfy the above four conditions related to give-right, give-left, take-right, and take-left for each cluster. By optimal we mean the minimum number of moves and minimum total moving distance. The following theorem shows that any violation of the conditions will result in the increase of the overall moving distance and/or the total number of moves to reach a load balance state.

Theorem 1. *Any violation of the four conditions on the give and take states of each cluster will result in the increase of the overall moving distance and/or the total number of moves to reach a load balance state.*

Proof. We consider four types of violations: take state changed to give state, give state changed to take state, take-right (take-left) changed to take-left (take-right), and give-right (give-left) changed to give-left (give-right).

Suppose cluster i's state is changed from take to give and one unit is sent to cluster j. To ensure load balancing, that one unit at cluster i will be compensated by another unit from cluster k (that is, k gives one unit

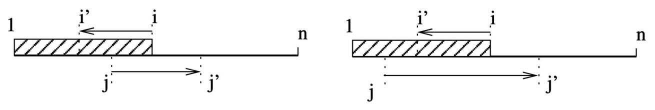

Fig. 4. Two cases for mixing up give-right with give-left.

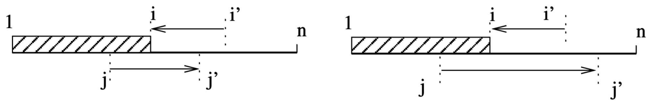

Fig. 5. Two cases for mixing up take-right with take-left.

back to i). A better scheme would be k giving one unit directly to j to save one move and shorten the distance if j and k are at the same side of i in the 1D array.

Suppose cluster i's state is changed from give to take and one unit is given from cluster j. To ensure load balancing, that one unit will be given away to cluster k. It would be better for j to give one unit directly to k to save one move and shorten the distance if k and j are at the same side of i.

When cluster i's state mixes give-right with give-left, we assume that one unit is moved from $w_i^{\rightarrow}$ to ${}^{\leftarrow}w_i$ (similarly, for ${}^{\leftarrow}w_i$ to $w_i^{\rightarrow}$). We show that this schedule will generate a longer moving distance. Suppose this unit is moved from i to i' $(1 \leq i' < i)$. Based on the balanced state requirement, one unit in cluster j in region $[1 \dots i-1]$ needs to be moved out to cluster j', with $i < j' \leq n$. We consider swapping these two units at i and j. To compare the moving distance between these two cases (before and after the swap), we consider two situations shown in Fig. 4 as follows:

1. When $i' \leq j < i$, we have
$$|i - i'| + |j - j'| > |j - i'| + |i - j'|.$$
2. When $1 \leq j < i'$, we have $|i - i'| + |j - j'| =$
$$|i - i'| + |j - i'| + |i' - j| > |j - i| + |i - j|.$$

In both cases, the moving distance before the swap $|i - i'| + |j - j'|$ is longer than that after the swap.

When cluster i's state mixes take-right with take-left, we again assume that one unit is moved from ${}^{\rightarrow}w_i$ to $w_i^{\leftarrow}$ (similarly, for $w_i^{\leftarrow}$ to ${}^{\rightarrow}w_i$). Suppose this unit is moved from i' to i $(i < i' \leq n)$. Based on the balanced state requirement, one unit in a cluster j in region $[1 \dots i-1]$ needs to be moved out to cluster j', with $n \geq j' > i$. We consider swapping these two units at i and j. To compare the moving distance between these two cases (before and after the swap), we consider two situations shown in Fig. 5 for $j' \leq i'$ and $j' > i'$. By following and argument similar to that in the above case, the moving distance before the swap $|i - i'| + |j - j'|$ is longer than that after the swap. □

The following theorem shows that, when the four conditions are met, the overall moving distance is independent of the actual schedule.

Theorem 2. *When take-right (take-left) states get loads from give-left (give-right) states, the overall moving distance is independent of the actual schedule.*

Proof. Let us consider schedules for all take-right states that get loads from give-left states. The case of the take-left states getting loads from give-right states can be argued in a similar way. Starting from cluster 1 and checking toward cluster n (that is, along the positive direction), for each unit of underload in a take-right state i, assign one unit of load from the closest give-left state i' (that is, a cluster in a give-left state with a minimum ID). Now, we show that all other assignments can be converted to the above schedule without changing the total moving distance. Suppose that, in the above state, the unit to i comes from a nonclosest give-left state j', and the unit from i' is assigned to a take-right state j, where $i \leq j \leq i'$. By swapping i' with j', the total moving distance remains the same, and the unit in i now comes from i' (see Fig. 6). This kind of swapping can be done iteratively. □

4.5 An Optimal Scan in 1D Arrays and Its Extension in 2D Meshes

In this section, we propose a simple sender-initiated optimal load balance algorithm for 1D arrays. The unique property is that the algorithm starts from each cluster in the give state (give-left and give-right) in parallel, without the need to be concerned with the detail of the take state of other clusters. Suppose i is in a take state, where $\bar{w} - w_i > 0$. Then, we do not distinguish take-right from take-left.

Sender-initiated optimal load balance in 1D arrays:

1. For each cluster i in the give state, the clusterhead sends $w_i^{\rightarrow}$ units to its right neighbor and sends ${}^{\leftarrow}w_i$ units to its left neighbor.
2. For each cluster i in the take state, when the clusterhead senses several bypassing units, it intersects as many units as possible to fill in its "holes." Unintersected units move along the same direction.

Theorem 3. *The proposed greedy schedule ensures an optimal schedule in 1D arrays.*

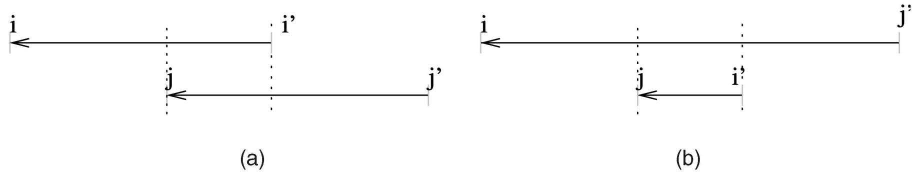

Fig. 6. Swapping of i' and j' without changing the total moving distance. (a) Before the swap. (b) After the swap.

Proof. It suffices to show that the case in Fig. 5 is avoided. That is, the two conditions related to the take state are satisfied. Based on the algorithm, when a unit is passed to i from right to left as shown in Fig. 5, it is implied that subarray $[i \dots n]$ is in the overloaded state. Similarly, when a unit is passed to j' from left to right, the subarray $[1 \dots j']$ is in the overloaded state. Since $i < j'$, the array $[1 \dots n]$, as a whole, is overloaded, which corresponds to a contradiction. □

When the scan procedure is extended from 1D arrays to 2D meshes, the scan procedure is applied twice: once on all rows and once on all columns. This 2D scanning process represents the core of SMART. However, this approach is no longer optimal in 2D meshes. For example, 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 its rows will change the 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 its 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 four moves occur; however, the optimal solution requires only two moves from $M[2,2]$ to $M[1,2]$ directly.

Theorem 4. *The ratio between the 2D scan and the optimal solution in terms of the number of moves is bounded by 2.*

Proof. During the 2D scan, wasted moves occur during the first scan when a (global) underloaded cluster i moves the load to another (global) underloaded cluster j. Suppose L units of load are moved from i and j. L units of load for j are necessary, whereas L units for i are wasted units. A similar situation occurs when a (global) overloaded cluster i moves load to another (global) overloaded cluster j. In this case, L units for j are wasted, whereas L units for i are necessary. It is easy to follow that for each wasted move, there is a matching necessary move; therefore, the ratio is bounded by 2. □

4.6 Several Variations of SMART

In SMART, an "aggressive" approach is used, where a local "give" state in a row or column can be a global "take" state. To avoid this situation, a "conservative" approach can be used to decide on local "give" and "take" states based on global average information.

Besides the prefix sum of the first i grids in a row (or column) in the positive direction, that is, $v_i = \sum_{j=1}^{i} w_j$, 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 the total sum in the row (or column). The negative-direction 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 a third kind of average as $\overline{w_m} = |\overline{w_g} - \overline{w_l}|/2$, the mean of global and local balanced states. This average is to achieve a compromise between conservative and aggressive approaches.

The variation 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 is in the "give" state. When $w_i - \overline{w} < 0$, it is underloaded and is 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 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 a load in the positive direction (or simply give-right) and ${}^{\leftarrow}w_i$ for a load in the negative direction (give-left) are changed as follows: A grid is in the "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 the amount does not cause the underload of the current node. More formally, we have

$$w_i^{\rightarrow} = \min\{w_i - \overline{w}, \max\{\overline{v'_{i+1}} - v'_{i+1}, 0\}\}, \tag{5}$$

$$^{\leftarrow}w_i = \min\{(w_i - \overline{w}) - w_i^{\rightarrow}, \max\{(\overline{v_{i-1}} - v_{i-1}), 0\}\}. \tag{6}$$

The threshold-based scan approach:

1. If $\overline{w} \neq \overline{w_l}$, then determine global balanced value $\overline{w_g}$.
2. Perform a row scan followed by a column scan by using the selected $\overline{w}$.
3. If $\overline{w} \neq \overline{w_l}$, then repeat Step 2 by 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-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 global 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. We use $\mathrm{SMART}(g)$, $\mathrm{SMART}(l)$, and $\mathrm{SMART}(m, c)$ to represent the threshold-based scan that uses the global average, the local average (the original SMART), and the mean of the global and local averages, respectively. c in $\mathrm{SMART}(m, c)$ corresponds to the number of iterations of Step 2.

If the total number of sensors is unknown, more information propagation is necessary. After the last cluster of each row gets the total number in its row, one more scan is generated in the last column to achieve the global average. Then, a scan in the negative direction in the column is conducted to distribute the average to each row.

5 Extended SMART

5.1 Simple Solutions

The 2D scan discussed previously works only when there is no hole. Otherwise, certain rows and columns may not be connected. In the worst case, the 2D mesh may be disconnected. A preprocessing is needed to plant "seeds" in holes at each 1D scan, and these seeds will serve as clusterheads in these holes.

Planting seeds in holes in an asymptotically optimal way is a nontrivial task. Suppose we want to optimize the total moving distance, the number of moves, and the communication latency (where each sequential neighbor communication is considered one step). The total moving distance should be $O(n^2)$ (as in the case of the first row of Fig. 1), the number of moves should be $O(n)$, and the communication latency should be $O(n)$.

A conservative approach could be sending out one seed at a time to an adjacent empty cluster. This will work for the case of the third row of Fig. 1, where k is a number larger than 5, and the direction is from left to right. However, this approach does not work well for the case of the first row, since the frontier node, which is the clusterhead of the first nonempty cluster in the expansion direction, needs to communicate with the leftmost node after each probing and expansion. The corresponding communication latency is $2\sum_{i=1}^{n-1} i = O(n^2)$. Note that, if the moving distance is the dominating factor rather than the communication latency, this is still an acceptable solution.

In an aggressive approach, each cluster that has a sufficient number of sensors (seeds) can send out multiple seeds to cover the rest. This approach certainly works for the case of the first row but fails for the case of the third row. In this case, the total moving distance would be

$$(n-1)^2 + (n-3)^2 + \ldots + 3^2 + 1^2 = O(n^3),$$

since clusters in the give state can initiate the process simultaneously. Moreover, the number of moves is $(n-1) + (n-3) + \ldots + 3 + 1 = O(n^2)$.

The simple recursive doubling does not work for the case of the second row either, where the span of each expansion is doubled in the subsequent step. This is because $\log n$ expansions will incur at least an $n/2 \times \log n = O(n \log n)$ communication latency, assuming that the initial span is 1.

5.2 Optimal Seed Planting in 1D Arrays with Holes

We propose a solution for the hole issue that is asymptotically optimal for several parameters, including communication latency ($O(n)$), total moves ($O(n)$), and total moving distance ($O(n^2)$), assuming that each cluster knows only the state of its two neighbors through probing. It is also assumed that the sensor network is sufficiently dense such that global $\overline{w} \geq 2$ (that is, on the average, each cluster has two sensors). Later, we will resort to a slightly stronger condition when the solution is extended from 1D arrays to 2D meshes.

First, we give some notation used in the solution. A *segment* S_i is a maximum sequence of nonempty clusters. W_i is the summation of load in S_i, and C_i is the length of S_i. Now, we introduce two important concepts related to S_i:

- *Expansion level* L_i of S_i: $2^{L_i} \leq C_i < 2^{L_i+1}$.
- *Energy level* E_i of S_i: $E_i = W_i - C_i$.

The expansion level L_i determines spans of successive expansions $2^{L_i}, 2^{L_i+1}, 2^{L_i+2}, \ldots$, whereas the energy level E_i indicates the number of denotable sensors in the segment. E_i should be large enough to cover holes in each expansion; that is, $E_i \geq 2^{L_i+k}$ for the kth expansion, which is called the *expansion condition*. Any cluster that has more than one sensor is in a denotable state for providing seeds, even though the cluster may be in an underloaded state.

The solution is based on a recursive doubling of the span for each successive expansion until there is no

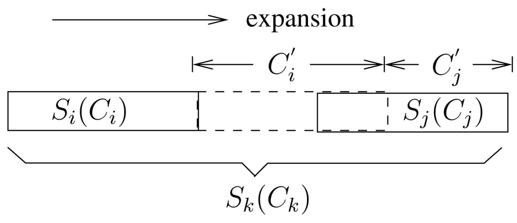

Fig. 7. The merging of two segments.

sufficient energy for expansion, but the actual size of expansion is governed by the current expansion level. For segment S_i with level L_i, the sequence of span is $2^{L_i}, 2^{L_i+1}, 2^{L_i+2}, \ldots$. For example, suppose the length C_i of S_i is 13. The first span is $2^3 = 8$, making a new segment with a length of 21; the next expansion with span $2^4 = 16$ will increase the length to 29, and so on.

Two approaches, reactive or proactive, can be used here. In the reactive approach, each cluster waits for an expansion signal from one of its predecessors or until a predefined timeout expires (the timeout value is given in Theorem 5). This approach trades a potential long delay for small total moving distance and total moves. This approach operates in the synchronized environment, where the synchronization point can be set during the initial deployment phase. In the proactive approach, each segment acts independently for expansion. This approach has a minimum communication latency but has occasional extra sensor movements for the lack of synchronization. The solution can be described in the following steps: 1) By following the positive direction, each segment performs an expansion through recursive doubling, at which point either it is informed from a predecessor segment or a predefined tim-out expires in the reactive approach, without waiting for any signal or timeout for activation in the proactive approach until it either reaches the last cluster of the 1D array or fails the expansion condition. 2) Repeat Step 1 for the negative direction, except that no timeout is needed at this step.

The efficiency of the method depends on the worst case time-out in the reactive approach and the excessive movement in parallel seed planting in the proactive approach. The next theorem shows that it is sufficient to set the timeout to $5(i-1)$, where i is the ID of the first cluster in the segment. The total moving distance in the proactive approach is still bounded within $O(n^2)$.

Theorem 5. *In each segment S in a scan, the total moving distance in constructing S is bounded by C^2 and the communication latency is bounded by $5C$.*

Proof. We prove by induction that, when S_i expands to connect S_j to form a new S_k along the positive direction, we assume that C_i' is the span S_i used to connect S_j, and C_j' is the span of the nonoverlapping region in S_j as in Fig. 7. Note that S_i may merge with another segment S_j to form a new segment S_k as a result of the expansion of S_i (as shown in Fig. 7). S_k will calculate its W_k and L_k accordingly. The special case S_j does not exist and has a length of 0. The following proof still applies.

Based on the induction, the latency in the formation of S_i is bounded by $5C_i$. In the current expansion, C_i is needed for the frontier node to inform all relevant clusters along the negative direction in S_i, and it takes $C_i + C'_i$ time to pass seeds to relevant positions. Finally, it takes C'_j steps to reach the frontier of S_k (that is, the rightmost node in S_j). Based on the fact that $C'_i \leq C_i < 2C'_i$ (expansion conditions), we have

$$5C_i + C_i + (C_i + C'_i) + C'_j < 5(C_i + C'_i + C'_j) = 5C_k.$$

Similarly, we show the total moving distance by induction. Based on induction, the formation of S_i is bounded by C_i^2. In the current expansion, the total moving distance is bounded by

$$\sum_{l=0}^{C'_i-1}(C_i + l) = C_iC'_i + C'_i(C'_i - 1)/2.$$

In the proactive approach, the formation of S_j needs to be included, which is bounded by $C_j^2 < (C'_i + C'_j)^2$. Based on the fact that $C'_i \leq C_i < 2C'_i$, we have

$$C_i^2 + C_iC'_i + C'_i(C'_i-1)/2 + (C'_i+C'_j)^2 < (C_i + C'_i + C'_j)^2 = C_k^2.$$

□

Since the method involves two sweeps, the overall moving distance is clearly bounded by $O(n^2)$, and the overall communication latency is bounded by $O(n)$. The total moves are bounded by $O(n)$ in the reactive approach and by $O(n \log n)$ in the proactive approach. In the latter case, clusters can plant seeds in parallel, but recursive doubling limits parallel merging to $\log n$ levels of the merging tree. Therefore, the proposed method in the proactive mode is optimal for the three parameters.

The following theorem shows that no timeout is needed in the second scan and proves the correctness of the 1D scan approach. The postfix of the 1D array is a subarray that contains the last cluster in the array.

Theorem 6. *Assume that the average load is at least 2 for each cluster. After the first scan, at least one postfix of the 1D array is a segment. In the second scan, no timeout is needed. All holes will be filled.*

Proof. It is assumed that the average load for each cluster is at least 2. Suppose $S_1, S_2, \ldots, S_{k-1}, S_k$ is the sequence of segments after Step 1 of preprocessing, where for each S_i (except S_k), $E_i < 2^{L_i}$; that is, $W_i < 2C_i$. If we let $\sum_{i=1}^{k-1} W_i = W_M$ and $\sum_{i=1}^{k-1} C_i = C_M$, then $W_M < 2C_M$. Based on the assumption of "at least average load of 2" for each cluster, we have

$$W_M + W_k \geq 2C_M + 2C_k > W_M + 2C_k;$$

therefore, $W_k > 2C_k$. S_k has sufficient energy for expansion. The only case for preventing such an expansion is when S_k includes the last cluster in the 1D array. Therefore, S_k is a postfix of the 1D array.

During Step 2 of preprocessing, since S_k has sufficient energy, it will fill in the "gap" (a consecutive sequence of empty clusters) between S_k and S_{k-1} by planting seeds in holes between them. By following the same argument, the newly formed segment will have sufficient energy to fill the next gap. In this way, all gaps will be filled after the second scan. □

The result from Theorem 6 shows that the scan process can be combined with the preprocessing (planting the seeds). That is, the scan process can start at step 2 of the preprocessing.

5.3 Extended SMART

Now, let us extend the approach from 1D to 2D. The first issue is to ensure that each 1D row array in the 2D mesh meets $\overline{w} \geq 2$. Instead of enforcing it (which is impossible), we propose a smoothing process on all columns before the preprocessing on rows. The smoothing process on columns includes preprocessing (that is, plant seeds in holes) and scan (that is, load balance). This columnwise smoothing process does not completely remove holes or balance the load along columns, unless the number of sensors in each column is at least $2n$ initially. However, when the sensor network is sufficiently dense, each row will have $\overline{w} \geq 2$ after the columnwise smoothing process. The following theorem shows the density requirement.

Theorem 7. *Suppose the average number of sensors in a cluster is at least 4. After the columnwise smoothing, each row will have at least $2n$ sensors.*

Proof. We try to find the maximum number of sensors that can be deployed when at least one row still has less than $2n$ sensors after the columnwise smoothing. If that number is less than $4n^2$, then the theorem is proven.

Assume that, initially, k columns have a load of at least $2n$, and the remaining $n - k$ columns have a load under $2n$. The former k columns will achieve load balancing after smoothing, whereas the latter $n - k$ columns will not. Without loss of generality, we assume that row 1 (that is, the first nodes in all columns) has less than $2n$ sensors after smoothing. All the first nodes of those $n - k$ columns that have not achieved the balanced state are holes. The maximum total load of nodes other than the first nodes in these $n - k$ columns is bounded by $(n - k)(2n - 1)$. The loads of the first nodes of the other k columns that have achieved the balanced state along columns are assumed to be $i_1, i_2, \ldots, i_k$, respectively. Based on the balanced state definition, the maximum total load of nodes other than the first nodes in these k columns is bounded by $(n - 1)[(i_1 + 1) + (i_2 + 1) + \ldots (i_k + 1)]$. Therefore, the total number is bounded by

$$\begin{aligned}&I + (n-1)(I+k) + (n-k)(2n-1) \leq \\ &\quad (2n-1) + (n-1)(2n+k-1) + (n-k)(2n-1)\end{aligned}$$

since $I = i_1 + i_2 + \ldots + i_k \leq 2n - 1$. Clearly, the total number is bounded by $4n^2 - (2 + k)n < 4n^2$. This number is maximized when $k = 1$, and the corresponding distribution is shown in Fig. 8. □

With the above result, the extended SMART protocol can be resolved to the following steps:

- Step 1 (Columnwise smoothing). Preprocess on columns (positive direction). If the last cluster fails condition 1 (discussed below), then terminate the preprocessing; otherwise, simultaneously preprocess and scan on columns (negative direction). If the first cluster fails condition 2 (discussed below), terminate the preprocessing; otherwise, scan on columns (positive direction).
- Step 2 (Row-wise preprocessing and scanning). Preprocess on rows (positive), then simultaneously

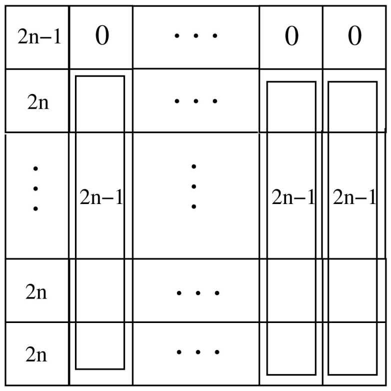

Fig. 8. A worst-case distribution.

preprocess and scan on rows (negative), and finally, scan on rows (positive).

- Step 3 (Columnwise scanning). Scan on columns (negative followed by positive).

Both conditions 1 and 2 are used for early termination when a particular column has less than $2n$ sensors. Condition 1 is defined when the last cluster is included in a segment S and $W \geq 2C$. Condition 2 is defined when the first cluster is included in a segment S such that $C = n$ and $W \geq 2n$. In Step 1, each column needs one, two, or three sweeps, depending on whether that column has $2n$ sensors or not. In Step 2, three sweeps are needed. Two sweeps are needed in Step 3. In the worst case, a total of eight sweeps are needed.

The above approach has potential drawbacks in generating a longer communication latency even in the absence of holes. To resolve this issue, we introduce some simple bookkeeping. Once the first sweep of Step 1 is completed, each end node in the last row will set a flag to 1 whenever it registers at least $2n$ sensors in the corresponding segment. If all flags in the last row are set, Step 3 can be skipped. Checking whether all flags are set can be done in parallel with Step 2, which needs $2n$ steps with two sweeps on the last row. The first sweep is a scan using Boolean AND, and the second is a broadcast of the scan value of the first sweep that is a Boolean value (1 for all flags set and 0 otherwise).

With the above modification, the worst case number of sweeps is reduced to 5. One more sweep can be eliminated by combining preprocessing and scanning in Step 1. Whenever the first cluster is included in the current segment, the scan process also starts. At the end of the first sweep, if the current segment includes both first and last clusters, then the third sweep in Step 1 can be eliminated, since its function can be done at the second sweep. The optimization for the number of moves discussed in Section 3 can still be used after the scan process starts. However, the number of moves during the smoothing and preprocessing phases cannot be further reduced.

6 Simulation

6.1 Simulation Environment

We use a custom simulator. The initial deployment that it generates could be a uniform or normal random distribution. We set up the simulation in a 500×500 area, which is the target field. The tunable parameters in our simulation are listed as follows: 1) *Cluster numbers* $n \times n$. A large n can improve the speed of deployment, whereas a small n can achieve more balanced results. We use 4 and 10 as n's values. 2) *Number of sensors* N. We have proven that at least $4n^2$ sensors are needed to guarantee the validation of SMART. Therefore, we vary N's value from 400 to 1,000. We also include cases of under $4n^2$ sensors to check the robustness of SMART. 3) *Normal distribution parameter* σ. σ is the standard deviation of the normal distribution of the initial deployment, which can control the density degree of the sensor clustering. We use 1 to 5 as its values. When σ is 1, around 98 percent of the sensors are in the 10 percent region of the area. When σ is 10, the distribution is very close to uniform random distribution. For each tunable parameter, the simulation is repeated 1,000 times. In addition to the proposed algorithms, we also simulate the traditional load balancing algorithms DIFF, EXCH, and the Voronoi-based localized sensor redistribution algorithm (VOR) for comparison.

The performance metrics are 1) deployment quality and 2) deployment cost. Deployment quality is shown by the balance degree measured by two simulation results. One is the *standard deviation* of the number of sensors in all the clusters. The other is *grads*, which is the difference between the largest cluster and the smallest one. Deployment cost is measured by the time of deployment, in terms of rounds, and energy consumption, in terms of overall moving distance.

6.2 Simulation Results

Fig. 9 compares the number of rounds and the moving distance of the three algorithms, namely, DIFF, EXCH, and SMART in uniform random distribution. Based on Figs. 9a, 9b, and 9c, we can see that the proposed SMART has a small and stable number of rounds. When the initial deployment is relatively balanced and n is small, every row can have more than $2n$ sensors; thus, it has five rounds; otherwise, it takes eight rounds (the worst case). DIFF and EXCH both have large numbers of rounds, which increase with the growth of node number, especially when n is large and the initial deployment is uneven. Figs. 9d, 9e, and 9f show the overall moving distance comparison. We can see that the overall sensor moving distance is proportional to the number of sensors. Therefore, the average moving distance of a sensor is insensitive to node numbers in all these algorithms. Among the three, SMART has the largest moving distance. This is because it achieves the most balanced final state, which leads to more sensor movements.

Figs. 10a and 10b show the balance degree of the results of these three algorithms by standard deviation in uniform random distribution. SMART achieves a balanced final state, and its standard deviation is less than or equal to 2. Figs. 10c and 10d show the balance degree in terms of grads. The grads of SMART are less than or equal to 2, and the grads in a row or a column are less than or equal to 1. In DIFF and EXCH, only the relative balanced state, which is the neighboring balance, is guaranteed. That is, the difference between adjacent clusters is less than or equal to 1. Therefore, the result could be a ladderlike distribution,

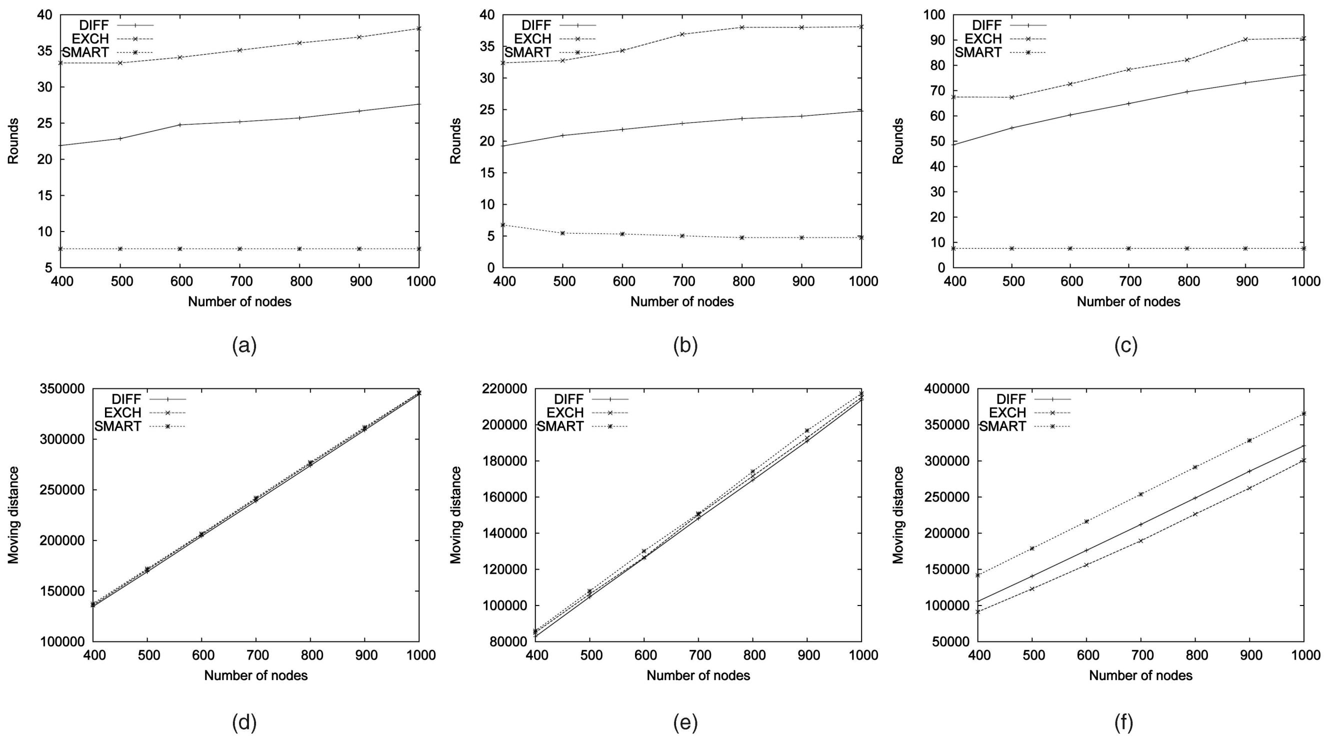

Fig. 9. Comparison of DIFF, EXCH, and SMART in (a), (b), and (c) round number and (d), (e), and (f) distance. (a) $n = 4$, $\sigma = 1$. (b) $n = 4$, $\sigma = 5$. (c) $n = 10$, $\sigma = 1$. (d) $n = 4$, $\sigma = 1$. (e) $n = 4$, $\sigma = 5$. (f) $n = 10$, $\sigma = 1$.

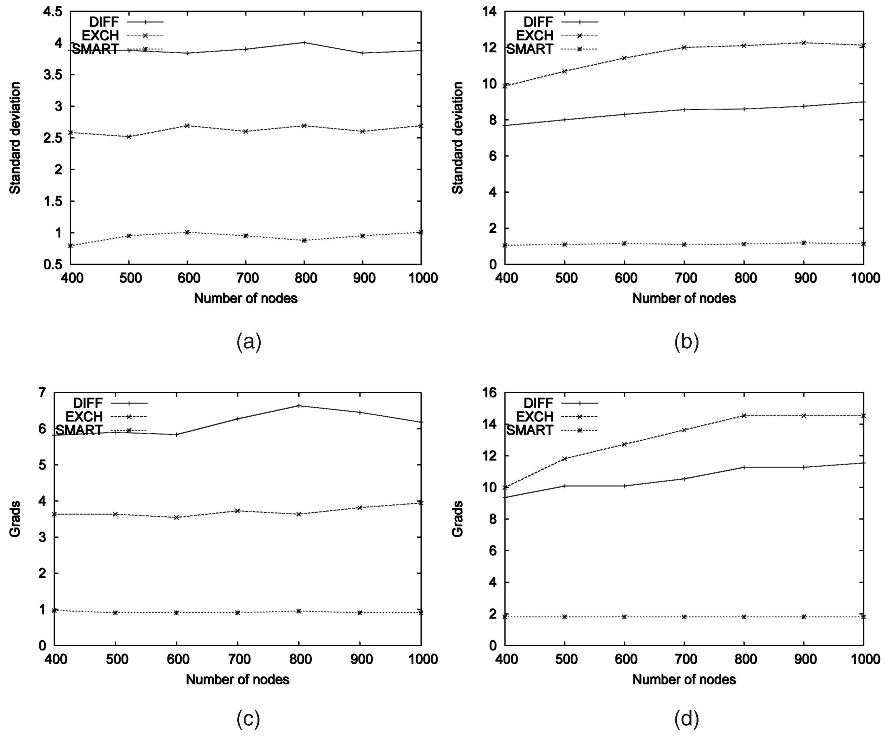

Fig. 10. Balance degree of DIFF, EXCH, and SMART $(\sigma = 1)$. (a) Standard deviation, $(n = 4)$. (b) Standard deviation, $(n = 10)$. (c) Grads, $(n = 4)$. (d) Grads, $(n = 10)$.

which leads to very large grads and standard deviations. When n is large, the grads of DIFF and EXCH are large, and their balance degrees are low.

Figs. 11a, 11b, 11d, and 11e compare the standard deviation and moving distance of algorithms using different normal distribution parameters σ. The curve "Initial" is the standard deviation of the initial deployment. SMART can achieve a more balanced state than DIFF and EXCH. SMART also outperforms them in a number of moves. In SMART, sensors move at most twice: one move for the vertical

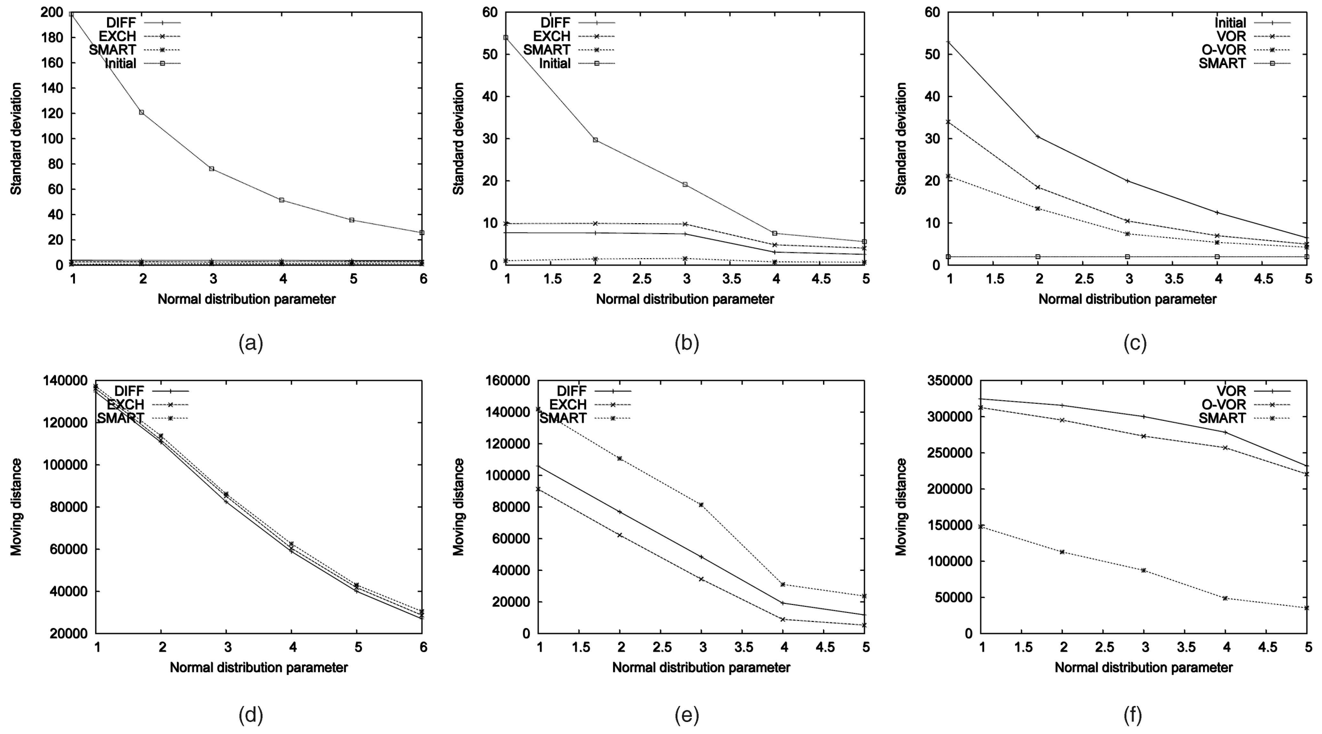

Fig. 11. SMART compared with DIFF, EXCH, and VOR by using different σ ($N = 400$). (a) Standard deviation, $n = 4$. (b) Standard deviation, $n = 10$. (c) Standard deviation, $n = 10$. (d) Moving distance, $n = 4$. (e) Moving distance, $n = 10$. (f) Moving distance, $n = 10$.

direction and another for the horizontal direction; more than 75 percent of sensors move only once. When N is 400, and σ is 1, SMART has 444, DIFF has 1,040, and EXCH has 1,137. Since the startup usually consumes more power than moving with invariable speed, less movement is desired. Fig. 11c shows the standard deviation, and Fig. 11f shows the moving distance comparison of VOR and SMART. We can see that VOR can only slightly reduce the standard deviation of the initial deployment. It has been mentioned in [5] that the basic VOR algorithm has difficulties in dealing with high-degree clustering, where sensors are centered around a few locations. When σ is 1, after applying VOR, the clustering area still has high density, whereas the original blank area has low density. The optimized VOR (O-VOR) proposed to deal with this problem is better than VOR, but SMART still outperforms O-VOR.

VOR is designed for a relatively sparse sensor network that has a uniform random initial deployment, whereas SMART is designed for a relatively dense network with high-degree clustering. For fairness, we conduct the following simulation to compare the performance of SMART and VOR in a relatively sparse network where the condition of Theorem 7 for SMART is not necessarily satisfied.

Figs. 12a and 12b show the comparisons of the resultant balance degree (in terms of standard deviation) and the number of rounds of SMART, VOR, and O-VOR ($\sigma = 3$, $n = 10$). In Fig. 12a, when N is larger than 400, SMART guarantees the balanced final state, where the standard deviation of the resultant deployment of SMART should be less than 2. This result is consistent with the analytical results in the previous section, where, if the average number of sensors in a cluster is less than 4, some rows may have less than $2n$ sensors after smoothing. When the node number is smaller than 400, the standard deviation is larger than 2, and the balanced status is not achieved. However, the increase of standard deviation is small, and the balance degree of SMART can still beat that of VOR. For VOR, when the node number is small, the resultant deployment is more balanced. With the growth of the number of deployed nodes, the balance degree gets lower. This is because, in the high-degree clustering environment, when the coverage termination condition of VOR is met, most of the area can be covered by at least one node, but VOR terminates before nodes in the clustering area scatter out. Fig. 12b shows the comparison of the number of rounds. At least 400 deployed nodes are needed to achieve the best performance, five rounds, for SMART. The worst is eight rounds. For VOR, a smaller node number leads to fewer rounds. However, VOR has fewer rounds than SMART when the node number is smaller than 150. O-VOR achieves a more balanced degree with a smaller round number than VOR.

Figs. 12c and 12d show the comparisons of several variations of SMART and also the optimal Hungarian-based method (OPT) in uniform and normal random distributions, respectively ($n = 10$, $N = 500$). $\mathrm{SMART}(l)$, $\mathrm{SMART}(g)$, and $\mathrm{SMART}(m, 3)$ are simulated. To check the effect of Step 3 in the threshold-based scan algorithm, we simulate $\mathrm{SMART}(g')$, which is $\mathrm{SMART}(g)$ without Step 3. As shown in Fig. 12c, $\mathrm{SMART}(m)$ has the most moving distance, whereas $\mathrm{SMART}(g)$ has a smaller moving distance than $\mathrm{SMART}(l)$. OPT has the smallest moving distance. Fig. 12d shows the results in normal random distribution. With the growth of σ, the moving

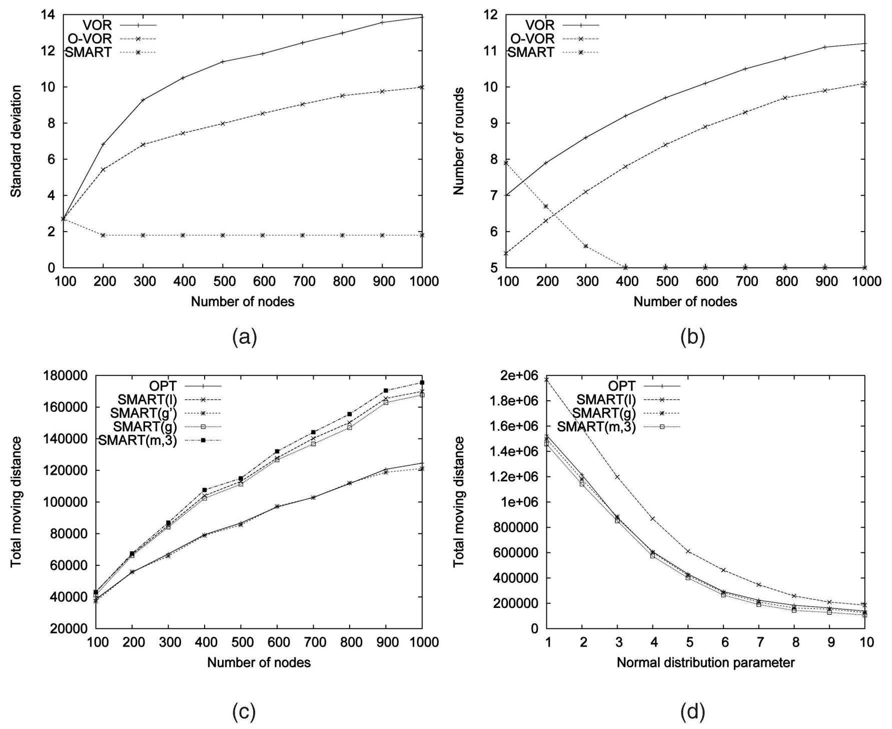

Fig. 12. Property analysis of SMART and VOR and comparison of variations of SMART. (a) Standard deviation. (b) Round number. (c) Moving distance, uniform. (d) Moving distance, normal.

distance decreases, and the number of moves decreases slightly. $\mathrm{SMART}(g)$ and $\mathrm{SMART}(m)$ have smaller moving distances than $\mathrm{SMART}(l)$. $\mathrm{SMART}(m)$ has the smallest among the three. $\mathrm{SMART}(l)$ has close or even better performance than OPT because it does not achieve a balanced result as OPT does.

Simulation results can be summarized as follows:

1. SMART achieves a more balanced state than DIFF, EXCH, and VOR sensor deployment methods in unevenly deployed sensor networks.
2. SMART needs a few rounds, which are bounded by eight, for load balancing.
3. The centralized optimal algorithm has the best performance; among all variations of SMART, $\mathrm{SMART}(g)$ has the best overall performance.
4. SMART can be effective when used in relatively dense sensor networks as a complement for the existing sensor deployment methods.
5. When the number of deployed nodes is less than $4n^2$, the performance of SMART is reduced, since more rounds are needed, and a balanced final state cannot be achieved.
6. In a sparse network, SMART may need more rounds than VOR to achieve a balanced degree, but it still beats VOR in terms of standard deviation.

7 Conclusion

In this paper, we have proposed the SMART algorithm, which is a hybrid approach of the local and global methods. We have considered a unique issue called the "communication hole," where certain sensing areas have no deployed sensors. A method of seed planting has been proposed to move one sensor to each uncovered area before the scanning process. We also developed an optimal solution that is based on the Hungarian method. Simulation results show that the proposed method can achieve an even deployment of sensors with modest costs. In the future, we will perform an in-depth simulation on energy consumption of sensor deployment algorithms and design some intracluster balancing algorithms to achieve high-resolution load balancing. We also plan to consider the case where only parts of the sensors are mobile. In this case, the ultimate goal is to maximize the minimum load of these grids. This is a more general measurement for the balance degree of the final distribution.

Acknowledgments

This work was supported in part by US National Science Foundation grants ANI 0073736, CCR 0329741, CNS 0422762, CNS 043533, EIA 0130806, CNS 0531410, and CNS 0626240, National Natural Science Foundation of China grants No. 60628201, No. 60473092, and No. 90612018, and National Basic Research Program of China grant No. 2006CB303000.

References

[1] I.F. Akyildiz, W. Su, Y. Sankrasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *IEEE Comm. Magazine,* pp. 102-114, Aug. 2002.
[2] D.E. Culler and W. Hong, "Wireless Sensor Networks," *Comm. ACM,* vol. 47, no. 6, pp. 30-33, June 2004.
[3] G.T. Sibley, M.H. Rahimi, and G.S. Sukhatme, "Robomote: A Tiny Mobile Robot Platform for Large-Scale Sensor Networks," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA),* 2002.
[4] O. Khatib, "Real Time Obstacle Avoidance for Manipulators and Mobile Robots," *Int'l J. Robotics Research,* vol. 5, no. 1, pp. 90-98, Aug. 1986.

[5] G. Wang, G. Cao, and T. La Porta, "Movement-Assisted Sensor Deployment," *Proc. IEEE INFOCOM,* Mar. 2004.
[6] Y. Zou and K. Chakrabarty, "Sensor Deployment and Target Localization Based on Virtual Forces," *Proc. IEEE INFOCOM,* Mar. 2003.
[7] M. Locateli and U. Raber, "Packing Equal Circles in a Square: A Deterministic Global Optimization Approach," *Discrete Applied Math.,* vol. 122, pp. 139-166, Oct. 2002.
[8] 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,* Sept. 2002.
[9] G.E. Blelloch, "Scans as Primitive Parallel Operations," *IEEE Trans. Computers,* vol. 38, no. 11, pp. 1526-1538, Nov. 1989.
[10] J. Albowicz, A. Chen, and L. Zhang, "Recursive Position Estimation in Sensor Networks," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* pp. 35-41,
[11] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-Less Low Cost Outdoor Localization for Very Small Devices," *IEEE Personal Comm.: Special Issue on Smart Spaces and Environment,* vol. 7, no. 5, pp. 28-34, Oct. 2000.
[12] A. Howard, M.J. Mataric, and G.S. Sukhatme, "Relaxation on a Mesh: A Formation for Generalized Localization," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS),* 2001.
[13] T.L. Casavant and J.G. Kuhl, "A Communication Finite Automata Approach to Modeling Distributed Computation and Its Application to Distributed Decision Making," *IEEE Trans. Computers,* vol. 39, no. 5, pp. 628-639, May 1990.
[14] G. Cybenko, "Load Balancing for Distributed Memory Multiprocessors," *J. Parallel and Distributed Computing,* vol. 7, pp. 279-301, 1989.
[15] H.C. Lin and C.S. Raghavendra, "A Dynamic Load Balancing Policy with a Central Job Dispatcher (LBC)," *IEEE Trans. Software Eng.,* vol. 18, no. 2, pp. 148-158, Feb. 1992.
[16] L.M. Ni, C.W. Xu, and T.B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," *IEEE Trans. Software Eng.,* vol. 11, no. 10, pp. 1153-1161, Oct. 1985.
[17] C.Z. Xu and F.C.M. Lau, *Load Balancing in Parallel Computers.* Kluwer Academic, 1997.
[18] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan, and K.K. Saluja, "Sensor Deployment Strategy for Target Detection," *Proc. Workshop Wireless Sensor Networks and Applications (WSNA),* 2002.
[19] S. Dhillon, K. Chakrabarty, and S. Iyengar, "Sensor Placement for Grid Coverage under Imprecise Detections," *Proc. Int'l Conf. Information Fusion,* 2002.
[20] S. Meguerdichian, F. Koushanfar, G. Qu, and M. Potkonjak, "Exposure in Wireless Ad Hoc Sensor Networks," *Proc. MobiCom,* 2001.
[21] D. Du, F. Hwang, and S. Fortune, "Voronoi Diagrams and Delaunay Triangulations," *Euclidean Geometry and Computers,* 1992.
[22] G. Wang, G. Cao, T. La Porta, and W. Zhang, "Sensor Relocation in Mobile Sensor Networks," *Proc. IEEE INFOCOM,* 2005.
[23] G. Wang, G. Cao, and T. La Porta, "Movement-Assisted Sensor Deployment," *IEEE Trans. Mobile Computing,* vol. 5, no. 6, pp. 640-652, 2006.
[24] http://www.darpa.mil/ato/progarms/shm/index.html, 2007.
[25] S. Chellappan, X. Bai, B. Ma, and D. Xuan, "Mobility Limited Flip-Based Sensor Networks Deployment," *Proc. IEEE Mobile Ad Hoc and Sensor Systems (MASS),* 2005.
[26] "Dictionary of Algorithms and Data Structures," 2005, http://www.nist.gov/dads/HTML/munkresAssignment.html.
[27] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization, Algorithms and Complexity.* Dover, 1998.
[28] O. Younis and S. Fahmy, "Distributed Clustering in Ad Hoc Sensor Networks: A Hybrid Energy-Efficient Approach," *Proc. IEEE INFOCOM,* Mar. 2004.

Shuhui Yang received the BS degree in computer science from Jiangsu University, Zhenjiang, China, in 2000 and the MS degree in computer science from Nanjing University, Nanjing, China, in 2003. She is a PhD candidate at the Department of Computer Science and Engineering, Florida Atlantic University. Her current research focuses on the design of localized routing algorithms in the wireless ad hoc and sensor networks.

Minglu Li graduated from the School of Electronic Technology, University of Information Engineering, in 1985 and received the PhD degree in computer software from Shanghai Jiao Tong University (SJTU) in 1996. He is a full professor and the vice chair of the Department of Computer Science and Engineering and the director of Grid Computing Center of SJTU. Currently, his research interests include grid computing, services computing, and pervasive computing. He has presided over 20 projects supported by the National Natural Science Foundation, National Key Technology R&D Program, 863 Program, 973 Program, and Science and Technology Commission of Shanghai Municipality (STCSM). He has published more than 100 papers in academic journals and international conferences. He is also a member of the Expert Committee of the ChinaGrid Program of Ministry of Education, a principal scientist of ShanghaiGrid, which is a grand project of STCSM, the Executive Committee of the ChinaGrid Forum, and the Executive Committee of the Technical Committee on Services Computing of the IEEE Computer Society.

Jie Wu is a distinguished professor at the Department of Computer Science and Engineering, Florida Atlantic University, and a program director at the US National Science Foundation. He is the editor of the text *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks* and is currently on the editorial board of several international journals. He served as the program vice chair for the 2000 International Conference on Parallel Processing (ICPP), the program vice chair for the 2001 IEEE International Conference on Distributed Computing Systems (ICDCS), and the general chair for the 2007 IEEE International Parallel and Distributed Processing Symposium (IPDPS). He was a program cochair of the IEEE First International Conference on Mobile Ad Hoc and Sensor Systems (MASS '04). He was a co-guest editor of a special issue of *Computer* on "Ad Hoc Networks." He also edited several special issues of the *Journal of Parallel and Distributing Computing* (JPDC) and the *IEEE Transactions on Parallel and Distributed Systems* (TPDS). He served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. His research interests are in the area of mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. He has published more than 300 papers in various journals and conference proceedings. He is the author of the text *Distributed System Design*. He is the recipient of the 1996–1997, 2001–2002, and 2006-2007 Researcher of the Year Award at Florida Atlantic University. He served as an IEEE Computer Society Distinguished Visitor and is the chairman of the IEEE Technical Committee on Distributed Processing (TCDP).