Deploying Wireless Sensor Networks under Limited Mobility Constraints

Sriram Chellappan, Wenjun Gu, Xiaole Bai, Bin Ma, Dong Xuan and Kaizhong Zhang

Abstract

In this paper, we study deployment of sensor networks using limited mobility sensors, where the maximum distance that sensors are capable of moving to is limited. Given an initial deployment of limited mobility sensors in a field that is clustered into multiple regions, our deployment problem is to determine a movement plan for the sensors to minimize the variance in number of sensors among the regions, and simultaneously minimize the sensor movements. Our methodology to solve this problem is to transfer the non-linear variance/movement minimization problem into a linear optimization problem through appropriate weight assignments to the regions. In our methodology, the regions are assigned weights corresponding to the number of sensors needed. During sensor movements across the regions, larger weight regions are given higher priority compared to smaller weight regions, while simultaneously ensuring minimum number of sensor movements. Following the above methodology, we propose a set of algorithms to our deployment problem. Our first algorithm is called the Optimal Maximum Flow based centralized (*OMF*) algorithm. In this algorithm, the optimal movement plan for sensors is obtained based on determining the minimum cost maximum weighted flow to the regions in the sensor network. We then propose an alternate algorithm called the Domain-based *OMF* (*D-OMF*) algorithm extending from the *OMF* algorithm that trades optimality with computational complexity and messaging overhead. Finally, we propose the Simple Pit-Peak based distributed (*SPP*) algorithm that uses local requests and responses for sensor movements. Using extensive simulations, we demonstrate the effectiveness of our algorithms from the perspective of variance minimization, number of sensor movements and messaging overhead under different initial deployment scenarios.

Index Terms

Sensor Networks, Deployment, Limited Mobility Sensors

Sriram Chellappan, Wenjun Gu, Xiaole Bai and Dong Xuan are with The Department of Computer Science and Engineering, The Ohio-State University, Columbus, OH 43210, U.S.A. E-mail: {chellapp, gu, baixia, xuan}@cse.ohio-state.edu. Bin Ma and Kaizhong Zhang are with The Department of Computer Science, University of Western Ontario, London, ON N6A5B7, Canada. Email: {bma, kzhang}@csd.uwo.ca.

I. INTRODUCTION

Mobility assisted sensor networks deployment has received significant attention recently [1], [2], [3] and [4]. In this realm, after initial deployment, sensors detect lack of desired deployment objectives. The sensors then estimate new locations to move to, and make the resulting movement. Since mobility is a power consuming operation, apart from deployment objective, a related objective in the above works is to also reduce the overall movement distance of sensors.

While the deployment objectives and corresponding methodologies are different in each of the above works, one assumption is that sensors are capable of unlimited movement distance. Specifically, if a sensor chooses to move to a desired location, it can do so without limitations in the movement distance. However, this may not be always feasible. For example, consider sensors that are battery powered. Clearly, the available power is a limited resource, and has to be shared for sensing, data-processing, transmission etc. Since mobility is a power consuming operation, the overall movement distance of the sensors thus cannot be unlimited. In some cases, external launching agents (that contain sensors inside them) are deployed in a field. Here, the mobility in sensors is realized by means of the external agents *launching out* sensors to new locations in the deployment field. Although in the case of launchers, the mobility does not depend on the battery power, the restriction on the mechanical features of the launchers will limit the movement distance. The launching process can be accomplished by means of spring activation, fuels and propellers etc. The limited mobility constraint means that; even if a sensor wants to move to a new location, it may be unable to do so if the distance to be traversed is beyond the movement capability of the sensor.

As a validation of our above claim, we briefly discuss two limited mobility sensor models already designed and realized in practice. Lymberopoulos and Savvides in [5] have recently designed a motion-enabled and power aware sensor node platform. The motion capability for the sensors comes from a battery enabled miniature geared motor that actuates the motion. The maximum movement distance of sensors in this design was determined as $165\ meters$. In another development, as part of the self-healing minefield program, DARPA has already designed a class of sensors with limited hop-by-hop mobility to detect and repair breaches in a battlefields [6]. Basically, the sensors are powered by fuels and a propeller, and the sensors can make up to about 100 hops. While the internal mobility semantics may be different in both sensor models, the fact is that the maximum movement distance is limited.

In this paper, we address an important sensor networks deployment problem under limited mobility

sensors. The sensor network we consider is a square field that has been clustered into multiple regions. The deployment objective is for each region to have a certain number of sensors, denoted by $\bar{k}$. At the time of initial deployment, not all regions will have $\bar{k}$ sensors. The sensors deployed are limitedly mobile. If a sensor moves from one region to any of its adjacent neighboring regions, we consider that as one hop made by the sensor. We denote H as the maximum number of such hops that a sensor is capable of making. In this context, our problem statement is; Given an initial deployment of sensors in the field, our problem is 1) to minimize the variance in the number of sensors among the regions in the network, and 2) simultaneously minimize the overall number of hops of the limited mobility sensors.

In this paper, we propose three algorithms to the above problem. The methodology of our algorithms is to transfer the non-linear variance/movement minimization problem into a linear optimization problem by means of weight assignments to the regions. By appropriately setting *weights* for the regions, we formally prove that maximizing sensor movements to regions with larger weights (under our weight assignment), will minimize the variance among the regions. The number of sensor movement hops is minimized by treating each sensor movement as a *cost*, and finding sensor movement paths with minimum costs.

Our first algorithm is called the Optimum Maximum Flow based centralized algorithm (*OMF* algorithm). In the *OMF* algorithm, the movement plan for sensors to optimize variance and sensor movement hops is obtained based on the *minimum cost maximum weighted flow* to the regions. Note that the *maximum weighted* flow problem is similar to the maximum flow problem except that each target (vertex) has a weight and the objective of the problem is to maximize the summation of the amount of flows through each target multiplied with the weight associated with each target. The maximum flow problem is its special case, where the weight of each target is one. In the *OMF* algorithm, we first construct a graph, called *virtual graph* (denoted by G_V) using global information on initial sensor network deployment, required number of sensors per region and sensor mobility capacity. In G_V, *sinks* for each region are created and appropriate *weights* are assigned to them. The maximum *weighted* flow to the multiple sinks, with minimum cost is determined in G_V. The corresponding flow plan in G_V is translated as a movement plan for sensors in the network. We subsequently prove the optimality of this movement plan in terms of minimizing variance and sensor movement hops.

We then propose an alternate algorithm, extending from the *OMF* algorithm. We call this algorithm as the Domain-based *OMF* (*D-OMF*) algorithm that trades optimality with global information exchange, computational complexity and messaging overhead. In the *D-OMF* algorithm, the sensor network is divided into domains, where each domain consists of multiple regions, and the *OMF* algorithm is executed

independently in each domain (without exchanging global information), and a movement plan for each domain is independently determined. Our third algorithm is called the Simple Pit-Peak based distributed algorithm (*SPP* algorithm). It is local, light-weight and purely distributed. In this algorithm, regions with less than the desired number of sensors locally send requests to their neighboring regions asking for sensors. The requests contain *weights* depending on number of sensors needed. Requests are served in a descending order of weights, along with minimizing sensor movement hops in serving them. Finally, we propose multiple approaches that can be used to combine and execute our above algorithms in practice, depending on application requirements and initial deployment scenarios.

We conduct an extensive performance comparison of our algorithms using simulations. We observe that in general the *OMF* algorithm (being optimal), achieves better variance minimization compared to the *D-OMF* and *SPP* algorithms. However, when sensors are deployed initially in groups throughout the network, the performance (variance minimization) of the *D-OMF* algorithm is close to that of the optimal *OMF* algorithm, if the size of the domain is carefully chosen. Also, under certain deployment conditions ($\bar{k}$ is small, uniform initial deployment), the performance of the *SPP* algorithm compares favorably with the *OMF* algorithm. We also study the overhead (in terms of sensor movements and messages) as a result of our algorithms. While the overhead in the *SPP* algorithm is generally lower than that of the *OMF* and *D-OMF* algorithms, we observe that when initial deployment is highly concentrated, the overhead in the *OMF* and *D-OMF* algorithms is close to the *SPP* algorithm, while even being smaller in some cases.

The rest of our work is organized as follows. In Section II, we formally define our deployment problem. In Section III we detail the methodology of our proposed algorithms. In Section IV, we present our *OMF* algorithm, and proofs of its optimality. In Section V, we present our *D-OMF* and *SPP* algorithms, and their features. Our performance evaluation is presented in Section VI. We provide some discussions in Section VII. Related work is presented in Section VIII, and we conclude our paper with some final remarks in Section IX.

II. Our Sensor Network Deployment Problem

A. Problem Definition

The sensor network we study is a square field of size Q. It is clustered into 2-dimensional square regions, where each region is of size R. The number of regions is denoted as S ($S = (\frac{Q}{R})^2$). We denote the number of sensors in region i at time of initial deployment as n_i. The deployment objective is for each region to have a certain number of sensors, denoted by $\bar{k}$. At the time of initial deployment, not

all regions will have $\bar{k}$ sensors. The sensors deployed are limitedly mobile. If a sensor moves from one region to any of its adjacent neighboring regions, we consider that as one hop made by the sensor. We denote H as the maximum number of such hops that a sensor is capable of making. In this context, our problem statement is; Given a sensor network with S regions each of size R, an initial deployment of N limited mobility sensors, our goal is to determine a sequence of sensor movements such that 1) at the conclusion of movements, the variance in the number of sensors from $\bar{k}$ among all the regions in the network with less than $\bar{k}$ sensors is minimized, and 2) the overall number of hops of the limited mobility sensors is simultaneously minimized. Denoting k_i as the number of sensors in a region i at the conclusion of sensor movements, the variance Var is,

$$Var = \frac{1}{S}\sum_{i=1}^{S}(\bar{k} - min(k_i, \bar{k}))^2. \tag{1}$$

Denoting h_i as the number of hops made by sensor i (where, $h_i \leq H$), and denoting N as the number of sensors initially deployed, the overall number of sensors movement hops is,

$$M = \sum_{i=1}^{N} h_i. \tag{2}$$

Our problem is to simultaneously minimize two objectives, namely Var (a non-linear function) and M.

Our problem is general, since we place no restriction on the value of $\bar{k}$. The value of $\bar{k}$ is application decided. If $\bar{k} = 1$, then it means a requirement of one sensor per region. For applications where redundancy is important (surveillance, mission critical, military applications), $\bar{k}$ can be set larger than 1. If $\bar{k}$ is a very large number, then our problem translates to load balancing the number of sensors among all regions in the network. An important feature of our problem is that we do not minimize the variance of number of sensors among all regions from $\bar{k}$. We minimize it among only the regions that have *less* than $\bar{k}$ sensors at final deployment, which is captured by the term $min(k_i, \bar{k})$ in equation (1). In many cases, sensors are *over-deployed*. When the deployment objective is only $\bar{k}$ sensors per region, the nature of our problem will not let extra sensors move, when the requirement of at least $\bar{k}$ sensors among all regions has been met. This is to preserve the mobility of sensors in such cases. Eventually, when some sensors fail (due to faults, power losses etc), the deficiency in $\bar{k}$ requirement can be met by the spare sensors whose limited mobility was initially preserved, effectively complementing the motivations for over-deployment.

We make the following assumptions in this paper. We assume that $min\{\frac{S_{sen}}{\sqrt{2}}, \frac{S_{tr}}{\sqrt{5}}\} \geq R$, where R is the region size, S_{sen} and S_{tr} are sensing and transmission ranges of the sensors respectively. If each region

has $\bar{k}$ sensors at final deployment, then the assumption $\frac{S_{sen}}{\sqrt{2}} \geq R$ implies that every point in each region is *covered* by $\bar{k}$ sensors, and the assumption $\frac{S_{tr}}{\sqrt{5}} \geq R$ implies that a sensor in any region can communicate with $\bar{k}$ sensors in each of its four adjacent regions. We assume that each sensor can know which region it resides in. To do so, sensors can be provisioned with GPS devices, or methods proposed in [7] can be applied, where location of sensors is determined by using sensors themselves as landmarks. For simplicity, we first assume that the regions to which a sensor can move to, are regions in its adjacent left, right, top and bottom directions only (denoted as neighboring regions). After discussing the above case for movement direction, the general case where a sensor can move in any arbitrary direction is discussed subsequently. Also, we assume that the network is not partitioned. The issue of partitions is discussed later.

B. An Example of our Problem and Challenges

We illustrate our problem further with an example. Consider an instance of initial deployment in the sensor network as shown in Figure 1 (a). The number inside circles denotes the number of sensors in that region. The number in the upper left corner denotes the corresponding region ID. Let maximum number of hops $H = 1$, and $\bar{k} = 2$. There are 32 sensors initially deployed. At the time of initial deployment, regions 2, 3, 9, 10, 11, 14, 15 have less than $\bar{k}$ sensors. An intuitive way to minimize the variance from $\bar{k}$ is to let neighboring regions locally synchronize for movement. Using local information exchanges, it is very likely that the sensors move according to the sequence shown in Figure 1 (a). The arrows indicate direction of movement, and the numbers beside arrows indicate number of sensors moved from the corresponding region.

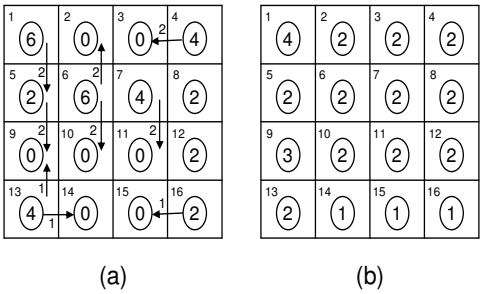

Fig. 1. An instance of initial deployment and an intuitive movement plan to minimize variance (a), and the resulting deployment (b).

Let us denote regions that have at least *one* sensor at initial deployment as *source regions* (or *sources*). Let us denote regions that do not have any sensor at initial deployment as *holes*. Region 4 is a source and moves sensors to region 3, since region 3 is close to it, and needs sensors. With local information

exchange, region 7 will not move sensors to region 3, rather it will move sensors to region 11, after synchronizing with regions 4 and 6. Similarly, since region 13 has four sensors, and since regions 9 and 14 do not have any sensor, a sensor moves from region 13 to fill each of the regions 9 and 14. But since region 5 has two sensors, and it receives two sensors from region 1, two sensors move from region 5 to region 9. Other regions also follow the same intuition and synchronization to move sensors. The resulting deployment is shown in Figure 1 (b). Note that regions 14, 15 and 16 have only one sensor. In fact with this movement plan, minimum variance (equal to 0) can never be achieved. Consider region 14. The only way region 14 can get a sensor is from regions 13, 10 or 15. However, regions 10 and 15 initially did not have any sensor. Thus, no sensor can move to region 14 via regions 10 and 15 since H is 1. Similarly, no sensor can move to region 13 via region 9. Besides, region 13 has no extra sensor now. Consequently all paths to region 14 are *blocked* in this movement plan. A pertinent question to raise at this point is, whether there exists an optimal plan that can make the variance 0. If so, what is the plan, or more importantly, what are the challenges that need to be addressed in determining such a movement plan. We discuss both issues in detail below.

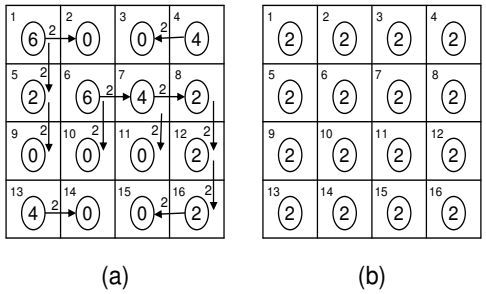

Fig. 2. An instance of initial deployment and an optimal movement plan (a), and the resulting deployment (b).

There are two key challenges to our problem. The first challenge is due to our objective of simultaneously minimizing variance and the number of sensor movement hops. Consider the movement plan in Figure 1 (a). Region 6 that has six sensors in it, wishes to fill regions 2 and 10. The intuition is because both regions are empty and region 6 is close to them. But this plan, that attempts to minimize hops, cannot minimize variance. There is thus a conflict that may be present in minimizing variance, and the number of hops using local information. For optimum deployment, region 6 should move sensors to regions 10 and 15 (in Figure 2 (a)). The path to region 15 may appear long, but it is the one that makes the global variance 0, shown in Figures 2 (a) and (b).

The second challenge is; due to limited mobility, if a sensor in one region wishes to move to some far away region, then depending on H, there must be mobile sensors in one or more intermediate regions

(like a chain) in the corresponding path (if $H = 1$, then all intermediate regions in the path need to have a mobile sensor). If there is no mobile sensor after a sensor has traveled H hops to a particular region, no sensor can move beyond that region, resulting in blocked paths. For instance in Figure 1 (b), although region 1 still has extra mobile sensors, all paths from region 1 to region 14 are blocked. The challenge is in determining such optimal chains for sensor movements. Such a path may traverse many intermediate regions as shown in Figure 2 (a), where the path from region 6 to region 15 traverses five regions, and a chain of movements is possible in each intermediate region. Determining such a chain of movements for optimal variance and sensor movement hops is not trivial. If sensors make purely local decisions, then optimality cannot be achieved. Also, it is preferable for sensors to make a movement plan (which sensors should move, and where) prior to their movement, in order to avoid erroneous movements, and compensating such errors later on.

III. Methodology of Our Algorithms

Our sensor network deployment problem has two objectives; 1) minimizing variance and 2) minimizing overall number of movement hops of the limited mobility sensors. In the following, we discuss our methodology to achieve both objectives. Consider any two regions i and j in a sensor network. Let the number of sensors in region i be less than that of region j, both of which are less than $\bar{k}$. If one sensor is available to move to one of these two regions, the contribution to global variance minimization in the network is larger if the sensor moves to region i than if it moves to region j. Our methodology to capture this notion of priority is by $weight$ assignment to regions. When sensors move, larger weight regions are given priority compared to smaller weight regions with the objective of global variance minimization. In the above example region i will have larger weight than region j to prioritize sensor movements to region i. We discuss our methodology in further detail below.

The overall variance is minimized (equal to 0), when each region has at least $\bar{k}$ sensors. Thus, for each region i in the sensor network, we first create $\bar{k}$ *virtual sinks* (or simply *sinks*) in order to allocate a position (*virtually*) for each of the $\bar{k}$ sensors that are needed in each region. Let each sink in region i be denoted by s_i^1, s_i^2, s_i^3, …, $s_i^{\bar{k}}$. For each sink s_i^1, s_i^2, s_i^3, …, $s_i^{\bar{k}}$, we assign weights to them denoted by w_i^1, w_i^2, w_i^3, …, $w_i^{\bar{k}}$ respectively to prioritize movements towards larger weight sinks. The weights for the sinks are given by,

$$w_i^j = 2 * j - 1 \quad (1 \leq j \leq \bar{k}). \tag{3}$$

Note that sink s_i^m has more weight than s_i^n, if $m > n$. Also note that $w_i^m = w_j^m$ for any two regions i

and j.

After sensors move towards sinks (according to their weights), some sinks will have sensors, while some do not. In order to capture the presence of a sensor in each sink among the multiple regions (after sensors move), we define the following function.

$$\phi_i^j = \begin{cases} 1, & if\ sink\ s_i^j\ has\ a\ sensor, \\ 0, & otherwise. \end{cases} \tag{4}$$

There is a constraint for the function ϕ. If $\phi_i^j = 1$, then $\phi_i^m = 1$ for all $m > j$. We are in effect saying here that if sink s_i^j in region i has a sensor, then each sink s_i^m in region i with larger weights (i.e., $m > j$) should have a sensor. The function ϕ captures whether a sink contains a sensor. We define a new metric here called *Score* as follows,

$$Score = \frac{1}{S}(\sum_{i=1}^{S}\sum_{j=1}^{\bar{k}} \phi_i^j \times w_i^j). \tag{5}$$

The *Score* function is the summation of weights of those sinks (for all regions) that contain a sensor in them. Clearly, the *Score* is larger when there are more sinks containing a sensor. The *Score* function also considers the weight of a sink. As such, in the event that a sensor can move to more than one sink, the *Score* is larger, when the sensor moves to the sink with the largest weight. Therefore during sensor movements, when we attempt to maximize the *Score*, we are in effect ensuring that as many sinks as possible contain a sensor, while also ensuring that larger weight sinks always have higher priority compared to smaller weight sinks. We now have the following Theorem.

Theorem 1: A sequence of sensor movements that maximizes $Score$ will minimize the variance Var and vice versa.

Proof: Consider two arbitrary sequences of sensor movements F and G, with functions $\{f_i^j\}$ and $\{g_i^j\}$ respectively. Assume there are m_i and n_i sinks in region i that have a sensor at the end of sequences F and G respectively. Recalling the constraint of ϕ_i^j in (4), we have,

$$f_i^j = \begin{cases} 1, & j > \bar{k} - m_i, \\ 0, & j \leq \bar{k} - m_i, \end{cases} \tag{6}$$

$$g_i^j = \begin{cases} 1, & j > \bar{k} - n_i, \\ 0, & j \leq \bar{k} - n_i. \end{cases} \tag{7}$$

The gain of $Score$ for sequence F compared with $Score$ for sequence G ($Score(F) - Score(G)$) is

$$= \frac{1}{S}\sum_{i=1}^{S}\sum_{j=1}^{\bar{k}}(f_i^j * w_i^j) - \frac{1}{S}\sum_{i=1}^{S}\sum_{j=1}^{\bar{k}}(g_i^j * w_i^j) = \frac{1}{S}\sum_{i=1}^{S}((m_i - n_i) * (2\bar{k} - m_i - n_i)). \quad (8)$$

The loss of variance Var of F compared with that of G ($Var(G) - Var(F)$) is

$$= \frac{1}{S}\sum_{i=1}^{S}(\bar{k} - n_i)^2 - \frac{1}{S}\sum_{i=1}^{S}(\bar{k} - m_i)^2 = \frac{1}{S}\sum_{i=1}^{S}((m_i - n_i) * (2\bar{k} - m_i - n_i)). \quad (9)$$

We can see that the amount of gain in $Score$ for F is the same as the amount of loss in Var. Thus, the sequence of sensor movements that maximizes $Score$ simultaneously minimizes Var, and vice versa.

■

From the above theorem, we can see that our original non-linear variance objective can be translated to a linear objective. In this paper, we propose three algorithms for our deployment problem, following the above methodology. In our algorithms, we create sinks for each region depending on the number of sensors needed. Each sink has a weight associated with it, such that when sensors move, sinks with larger weights have higher priority compared to sinks with smaller weights. The goal of our algorithms is to maximize *Score*, which according to Theorem 1 minimizes the variance Var.

The second objective of our problem is minimizing total number of sensor movement hops. We achieve this goal by treating sensor movement hops as costs, and minimizing such costs in our algorithms. When there are multiple sinks in other regions with same weights, our algorithms will ensure that sensors move to sinks in those regions that are closer in terms of distance to be traversed. Clearly, larger weight sinks are still given priority compared to smaller weight sinks. However, with such movements, the resulting number of overall sensor movement hops is minimized, along with maximizing *Score*. If a sensor in a region does not need to move to another region we treat the sensor as *virtually* moving to a sink in the same region. Such a movement incurs 0 cost.

IV. The Optimal Maximum Flow Based Centralized Algorithm

A. Overview

Our first algorithm is the *Optimal Maximum Flow based centralized algorithm* (*OMF* algorithm). In the *OMF* algorithm, the sensor network at initial deployment is translated as a graph structure. The algorithm then determines the minimum cost maximum weighted flow in the graph. The corresponding flow plan in the graph is translated as a movement plan for the sensors in the network. In the following, we describe

our *OMF* algorithm from the perspective of a Base-station executing the algorithm. An alternate approach to execute the *OMF* algorithm is presented in Section V-A.

Algorithm 1 Pseudocode of the *OMF* algorithm

1: Collect the information on the number of sensors in each region in the sensor network.
2: Construct a graph $G_V(V_V, E_V)$ using the above region information, desired number of sensors per region $\bar{k}$ and the sensor mobility capacity H. G_V models the sensor network at initial deployment time.
3: Determine the minimum cost maximum weighted flow from source regions to weighted sinks in G_V.
4: Determine a movement plan for the sensors in the sensor network based on the above flow plan in G_V.
5: Forward the movement plan to sensors in the network.

Algorithm 1 shows the sequence of steps in the *OMF* algorithm. In Step 1, each sensor in the network identifies which region it resides in. Sensors then forward information on the number of sensors in their region towards the Base-station. For routing packets towards Base-station, protocols like [8], [9], [10] can be used, where the protocols route packets towards intended destinations in the network (Base-station in our case) using shortest paths. The Base-station thus obtains information on the number of sensors in all regions in Step 1. As pointed out before, for determining which region a sensor resides in, sensors can be provisioned with GPS devices or methods proposed in [7] can be used where location of sensors is determined by using sensors themselves as landmarks. Also, we assume the network is connected without partitions. The issue of partitions is discussed later.

In Step 2, the Base-station constructs a *virtual graph* (G_V), which is a graph structure whose vertices and edges model the regions and sensor movement ability between regions respectively at initial deployment. In Step 3, the Base-station determines the maximum weighted flow to the sinks in G_V (that maximizes equation 5) with minimum cost. In Step 4, the flow plan in the G_V corresponding to the minimum cost maximum weighted flow is translated as a movement plan for the sensors. In Step 5, the Base-station forwards the movement plan (which sensors should move and where) to the sensors in the network. We subsequently prove that this movement plan minimizes the variance, and overall number of sensor movement hops in the sensor network. Each of Steps 2, 3, 4 and 5 in our *OMF* algorithm is discussed in detail below.

B. Constructing the Virtual Graph G_V

We now discuss Step 2, that involves the construction of the virtual graph denoted by $G_V(V_V, E_V)$. Before we discuss G_V, we introduce the notation of *reachability* between regions. For any region i in the sensor network, we denote its *reachable* regions as those regions to which a sensor from region i can move to. Obviously, the reachable regions depend on the maximum movement hops H. We first assume

that the regions to which a sensor can move to, are regions in its adjacent left, right, top and bottom directions only. Thus, if $H = 1$ in Figure 1, then the reachable regions for region 1 are regions 2 and 5. If $H = 2$, the reachable regions are regions 2, 3, 5, 6 and 9.

The construction of G_V involves 1) The establishing of vertices and edges for each region in the sensor network and creation of sinks for each region, 2) The establishing of reachability relationship between the regions, 3) Adding weights to sinks following our discussions in Section III and 4) Adding costs to edges to capture sensor movements across regions. The objective of this construction is to ensure that G_V models the sensor network, identifies sources, sinks, and reachability relationship among regions. Figure 3 (a) shows an instance on initial deployment for a 2×2 network with 4 regions and 12 sensors, and where $\bar{k} = 3$ and $H = 1$. Its corresponding virtual graph G_V is shown in Figure 3 (b). The numbers inside the circles in Figure 3 (a) denotes the number of sensors in the corresponding region in the sensor network. In the following, we describe the virtual graph construction process in detail. Let us first describe the establishment of vertices and edges assignment in G_V for one arbitrary region in the sensor network. Without loss of generality, consider region i with initially n_i sensors. For this region, we create a vertex called as the *base* vertex of region i (denoted by v_i^b) in G_V. We create vertex v_i^{out} to keep track of the number of sensors that can move out from region i. We then create $\bar{k}$ sink vertices for region i (due to deployment requirement of $\bar{k}$ sensors per region). The sink vertices for region i are denoted by vs_i^1, vs_i^2, vs_i^3, ..., $vs_i^{\bar{k}}$. We also create vertex v_i^{in} as a proxy for the $\bar{k}$ sink vertices.

The next step is adding edges between vertices for this region. An edge of capacity n_i is added from v_i^b to v_i^{out}. This means that up to n_i sensors can move from region i. Since v_i^{in} is a proxy for the sink vertices, the capacity from v_i^{out} to v_i^{in} is also n_i. From v_i^{in}, an edge is added to each of the vertices vs_i^1, vs_i^2, vs_i^3, ..., $vs_i^{\bar{k}}$ with capacity 1. Since the deployment requirement is $\bar{k}$ sensors per region, we allow up to *one* sensor to move to each sink (for $\bar{k}$ such sinks). All other regions are treated similarly in G_V. For example, for region 1 in G_V in Figure 3 (b), we create six vertices corresponding to the *base* vertex (v_1^b), *in* vertex (v_1^{in}), *out* vertex (v_1^{out}) and $\bar{k} = 3$ sink vertices (vs_1^1, vs_1^2 and vs_1^3). Edges between the vertices, and their capacities for region 1 are also shown. All other regions are treated similarly.

The second step is establishing reachability relationship among the regions into G_V. Let us consider two arbitrary regions i and j that are reachable from each other. In G_V, edges are added from v_i^{out} to v_j^{in}, with edge capacity n_i, which is the number of sensors in region i. This is to allow up to n_i sensors to move from region i to region j. Correspondingly, edges are added from v_j^{out} to v_i^{in} , with capacity n_j. For example in Figure 3 (b), there is an edge from v_1^{out} to v_2^{in} with capacity $n_1 = 4$, and an edge from

v_2^{out} to v_1^{in} with capacity $n_2 = 2$ since regions 1 and 2 are reachable from each other.

The next steps are weight assignment to sinks, and cost assignment to edges. Consider region i again. For sinks vs_i^1, vs_i^2, vs_i^3, ..., $vs_i^{\bar{k}}$ in region i, we denote their weights as w_i^1, w_i^2, w_i^3, ..., $w_i^{\bar{k}}$. Following from the discussions in Section III, the values for the weights are 1, 3, 5, ..., $2\bar{k}-1$ respectively. Since $\bar{k} = 3$ in the example in Figure 3, we have weights 1, 3 and 5 for the sinks (shown along side the sink vertices). Note that, w_i^m is larger than w_i^j, if $m > j$. We now discuss costs for edges between regions in G_V in order to capture number of sensor movements. If a sensor moves from its region to its adjacent region, then it denotes one hop made by the sensor. Let us consider two regions i and j in the sensor network that are reachable from each other. Let the distance between them in terms of number of hops be $d_{i,j}$. That is, $d_{i,j}$ denotes the minimum number of hops required for a sensor in region i to move to region j (or vice versa). For instance, in Figure 1 (a), $d_{1,3} = d_{3,1} = 1$. Obviously $d_{i,j} \leq H$, if regions i and j are reachable from each other. To incorporate this in G_V, between any two reachable regions i and j, the costs of edges from v_i^{out} to v_j^{in}, and the costs of edges from v_j^{out} to v_i^{in} are assigned as $d_{i,j}$. Apart from the above, the only remaining edges in G_V are the ones from v_i^b to v_i^{out}, from v_i^{out} to v_i^{in} , and from v_i^{in} to vs_i^1, vs_i^2, vs_i^3, ..., $vs_i^{\bar{k}}$ (for all regions i). These edges denote internal movements within a region, and the cost for these edges is set as 0. The costs of edges in G_V are not shown in Figure 3.

At this point, Step 2 of our *OMF* algorithm is completed. The Base-station has constructed G_V that models the sensor network at initial deployment. Before proceeding to Step 3, we define a flow plan Z in G_V and a metric W. Z is the sequence of flows (in G_V) that meets the following condition; $W = \sum_{i=1}^{S} \sum_{j=1}^{\bar{k}} (f_i^j * w_i^j)$ is maximized, where f_i^j is the subflow to sink vs_i^j in flow plan Z. We call Z as a maximum weighted flow plan in G_V. If the cost of Z is minimized, Z is called as a *minimum cost maximum weighted flow* plan. With sinks in G_V having weights associated with them, a maximum weighted flow plan must maximize the number of sink vertices that receive a flow, and prioritize flows to larger weight sinks first compared to smaller weight sinks in G_V. Since the capacity of the edge from v_i^{in} to vs_i^j in G_V is 1, f_i^j meets the constraint of function ϕ defined in (4). Since G_V is a translation of the sensor network, the flow plan Z in G_V can be translated as a corresponding movement plan for sensors in the sensor network (exactly how this is done is discussed in Section IV-D). From the definition of *Score* in (5) and W above, the corresponding sensor movement plan maximizes the *Score* with minimum cost, which in turn minimizes Var (from Theorem 1) with minimum cost. To summarize, with the construction of G_V in place, the variance/movement minimization problem now becomes one, where the weighted flow

to sinks in G_V is to be maximized with minimum cost.

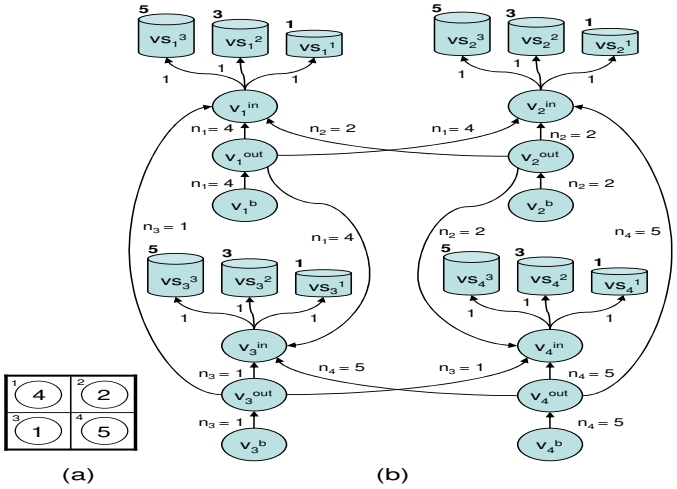

Fig. 3. An instance of the initial network deployment (a) and the corresponding virtual graph G_V (b).

C. Computing the Minimum Cost Maximum Weighted Flow in the Virtual Graph G_V

We now proceed to Step 3 in the *OMF* algorithm, where determine the minimum cost maximum weighted flow in G_V. In the following, we present our algorithm to determine the minimum cost maximum weighted flow in G_V. Our approach to maximize the weighted flow is to translate larger weight sink vertices, as lower cost sink edges. Thus, prioritizing flows to large weight sinks now becomes prioritizing flows through lower cost edges. This is the crux of our algorithm described below.

Algorithm 2 Pseudocode for computing the minimum cost maximum weighted flow in G_V

```
Input: G_V(V_V, E_V), H, S and k̄
Output: Graph G_V^m(V̄, Ē) and Minimum Cost Maximum Weighted Flow Plan Z in G_V
|E_V| = No. of Edges in G_V, |V_V| = No. of Vertices in G_V
|Ē| = |E_V| + S × (k̄ + 1)
|V̄| = |V_V| + 2
Add vertices S^v_source and S^v_sink to G_V to create graph G^m_V
for each region i do
    for j from 1 to k̄ do
        Add edge from sink vs_i^j to S^v_sink
        Assign corresponding edge capacity as 1
        Assign corresponding edge cost as −(2j − 1) × H × |Ē|
    end for
    Add edge from S^v_source to v_i^b
    Assign corresponding edge capacity as ∞
    Assign corresponding edge cost as 0
end for
Determine the maximum flow value |Z̄| from S^v_source to S^v_sink in G^m_V
Determine the minimum cost flow plan Z (for flow value |Z̄|) from S^v_source to S^v_sink in G^m_V
```

Algorithm 2 is the pseudocode to determine the minimum cost maximum weighted flow in the virtual graph G_V. The input is $G_V(V_V, E_V)$, H, number of regions S and $\bar{k}$. We first create a new graph from G_V called $G_V^m(\bar{V}, \bar{E})$ as follows. We first create two new vertices called *Super Source* and *Super Sink*,

denoted by S^v_{source} and S^v_{sink} respectively. Edges are added from each sink vertex to S^v_{sink}, with capacity 1 to allow only one sensor to move from each sink towards S^v_{sink}. The cost of the edges from sinks vs^1_i, vs^2_i, vs^3_i, ..., $vs^{\bar{k}}_i$ to S^v_{sink} (for all regions i) are set as $-H \times |\bar{E}|$, $-3 \times H \times |\bar{E}|$, $-5 \times H \times |\bar{E}|$, ..., $-(2\bar{k}-1) \times H \times |\bar{E}|$ respectively, where $|\bar{E}|$ is defined in Algorithm 2 [1]. Finally, edges are added from S^v_{source} to all base vertices (i.e., v^b_i for all i), with capacity ∞ to allow any amount of flow from S^v_{source}. The costs for these edges are set as 0, since the flow through such edges are not actual sensor movements.

At this point (Step 16 in Algorithm 2), G^m_V has been constructed. Determining the flow plan to maximize weighted flow to sinks with minimum cost in G^m_V is a two-step process (Steps 17 and 18 in Algorithm 2). The Base-station will first determine the maximum flow value ($|\bar{Z}|$) from S^v_{source} to S^v_{sink} in G^m_V. The maximum flow value $|\bar{Z}|$ indicates the maximum number of sinks that can get a sensor in G^m_V. However, this only indicates the maximum *number* of sinks. The determination of the maximum flow value does not consider the fact that sinks have different weights and larger weight sinks need to be accorded higher priority. Our objective however, is to determine the *flow plan* Z (the actual flow among the edges) in G^m_V such that *weighted* flow to sinks is maximized with minimum cost. We do this in Step 18 by determining the minimum cost flow plan Z (for maximum flow value $|\bar{Z}|$) in G^m_V, discussed further below.

We know that when executing the minimum cost flow algorithm on any graph, flow is prioritized through edges with lower cost. By setting the edge costs from sinks to S^v_{sink} as the negative of weights of the corresponding sink, we will achieve our objective of prioritizing flow to sinks with larger weights in determining the minimum cost flow to S^v_{sink}. There is one issue we have to resolve during cost assignment. Recall that sensor movements between reachable regions are considered as costs in G^m_V. Clearly, these costs will affect the minimum cost flow plan when determining flows to sinks with minimum cost in G^m_V. To prevent this from happening, the costs from sinks to S^v_{sink} is assigned as the negative of the sink weights multiplied by a large constant (namely, $H \times |\bar{E}|$). This constant is large enough to ensure that the flow plan (Z) to maximize weighted flow in G^m_V is not affected by the costs between reachable regions, while still minimizing costs between reachable regions (that denote sensor movements). Before discussing how to translate this flow plan Z into a sensor movement plan, we state the following theorem showing the relationship between G^m_V and G_V.

Theorem 2: The flow plan corresponding to the minimum cost maximum flow in G^m_V is the flow plan corresponding to the minimum cost maximum weighted flow in G_V.

Proof: We first prove that the flow plan corresponding to the minimum cost maximum flow in G^m_V

[1]The interpretation of $|\bar{E}|$ is discussed subsequently.

is the flow plan corresponding to the maximum weighted flow in G_V. We will prove this by contradiction. Let the minimum cost maximum flow plan in G_V^m be Z. Suppose Z does not yield the maximum weighted flow in G_V. This means there exists a flow plan Y that has a higher weighted flow than that of Z. Let us denote the weighted flow values of Z and Y to sinks in G_V by W_Z and W_Y respectively. We then have $W_Y - W_Z \geq 1$. Denoting $Cost_Z$ and $Cost_Y$ as the cost values of Z and Y in G_V^m respectively, we have,

$$Cost_Z = -W_Z * |\bar{E}| * H + Cost'_Z \quad (10)$$

$$Cost_Y = -W_Y * |\bar{E}| * H + Cost'_Y \quad (11)$$

in which $Cost'_Z$ and $Cost'_Y$ denote the sum of the edge costs from v_i^{out} to v_j^{in} for all regions i and j in Z and Y respectively. Since Z applies minimum cost flow algorithm, we have $Cost_Z < Cost_Y$. However, we can also obtain,

$$Cost_Z = -W_Z * |\bar{E}| * H + Cost'_Z \geq -W_Z * |\bar{E}| * H \geq -W_Y * |\bar{E}| * H + |\bar{E}| * H$$
$$> -W_Y * |\bar{E}| * H + Cost'_Y = Cost_Y,$$

which is a contradiction. Therefore, flow plan Z yields the maximum weighted flow in G_V. Since Z is the plan after executing the minimum cost algorithm in G_V^m, the costs of flow among edges between reachable regions is minimized in G_V^m. G_V is made of exactly the same edges (edges between reachable regions). Therefore, flow plan Z corresponds to the minimum cost maximum weighted flow in G_V. ■

D. Determining the optimal movement plan from the virtual graph G_V

Once the minimum cost maximum weighted flow to each sink in G_V (and the corresponding flow plan in all edges in G_V) is obtained, we proceed to Step 4 in Algorithm 1. In Step 4, we translate the flow plan from Step 3 into actual sensor movements as follows. Let Z^V denote the flow plan (a set of flows) corresponding to the minimum cost maximum weighted flow algorithm in G_V, where the capacity of each flow is 1. Each flow $z^V(v_i^b, vs_j^x) \in Z^V$ is a flow from v_i^b to vs_j^x in G_V. The flow $z^V(v_i^b, vs_j^x)$ is of the form $\langle v_i^b, v_i^{out}, v_j^{in}, vs_j^x \rangle$. Thus, for the flow plan Z^V, we can map it to a corresponding movement plan Z^S (set of movement sequences for sensors) in the sensor network. That is for each $z^V(v_i^b, vs_j^x)$ $(\in Z^V)$ of the form $\langle v_i^b, v_i^{out}, v_j^{in}, vs_j^x \rangle$, the corresponding $z^S(i,j)$ $(\in Z^S)$ is of the form $\langle i, j \rangle$. Physically, this means that one sensor should move from region i to region j. The sensor movement plan Z^S (consisting of the set of all such z^S, obtained from z^V) is our output. This movement plan that indicates which sensors should move, and their corresponding destinations are forwarded by the Base-station to the sensors in the network.

E. Optimality of our OMF *Algorithm*

Before discussing optimality of our algorithm, we first introduce the concept of feasible flows and movement sequences. We call a flow $z^V(v_i^b, vs_j^x)$ of the form $\langle v_i^b, v_i^{out}, v_j^{in}, vs_j^x, \rangle$ feasible in G_V if there exists positive edge capacities from vertices v_i^b to v_i^{out}, v_i^{out} to v_j^{in}, v_j^{in} to vs_j^x. We call a movement sequence $z^S(i,j)$ of the form $\langle i,j \rangle$ feasible in the sensor network if there is at least one mobile sensor in region i that can move to region j. We have the following lemma for a flow in G_V and a movement sequence in the sensor network.

Lemma 1: A flow $z^V(v_i^b, vs_j^x)$ in G_V is feasible if and only if the corresponding movement sequence $z^S(i,j)$ is feasible in the sensor network.

Proof: We first prove if $z^S(i,j)$ is feasible, then $z^V(v_i^b, vs_j^x)$ is feasible. If $z^S(i,j)$ is feasible, then there is at least one mobile sensor in region i, and regions i and j are reachable from each other. That is, the capacities of the edges from v_i^b to v_i^{out}, and from v_i^{out} to v_j^{in} are ≥ 1, and there exists an edge from v_j^{in} to vs_j^x, whose capacity is 1 (from Section IV-B). Thus, $z^V(v_i^b, vs_j^x)$ is feasible.

We now prove if $z^V(v_i^b, vs_j^x)$ is feasible, then $z^S(i,j)$ is feasible. If $z^V(v_i^b, vs_j^x) = \langle v_i^b, v_i^{out}, v_j^{in}, vs_j^x \rangle$ is feasible, then the capacities of the edges from v_i^b to v_i^{out}, from v_i^{out} to v_j^{in} and from v_j^{in} to vs_j^x are all ≥ 1. This implies that there is a sensor in region i, and regions i and j are reachable from each other. So a sensor can move from region i to region j. Thus $z^S(i,j)$ is feasible. ■

We obtain the following corollary from Lemma 1.

Corollary 1: For a feasible flow plan $\bar{Z}^V$ (set of all z^V) in G_V, a corresponding feasible sensor movement sequence plan $\bar{Z}^S$ (set of all z^S) can be found in the sensor network and vice versa.

Proof: We first prove that for a feasible flow plan $\bar{Z}^V$ in G_V, a corresponding sensor movement sequence plan can be found in the sensor network. Consider an arbitrary feasible flow z^V in $\bar{Z}^V$. By Lemma 1, a corresponding feasible sensor movement sequence z^S in the sensor network can be found for the flow z^V in G_V. The set of such sensor movement sequences is $\bar{Z}^S$ in the sensor network.

We now prove that for a feasible sensor movement sequence plan $\bar{Z}^S$ in the sensor network, a corresponding flow plan can be found in G_V. Consider an arbitrary feasible sensor movement sequence z^S in $\bar{Z}^S$. By Lemma 1, a corresponding feasible flow z^V in G_V can be found for the sensor movement sequence z^S in the sensor network. The set of such flows is $\bar{Z}^V$ in G_V.

■

The following Theorem shows that the movement plan obtained by our *OMF* algorithm optimizes both variance and the number of sensor movement hops.

Theorem 3: Let Z^V_{opt} be the minimum cost maximum weighted flow plan in G_V. Its corresponding movement plan Z^S_{opt} will minimize variance and the number of sensor movement hops in the sensor network.

Proof: We first prove that our *OMF* algorithm is optimal in terms of minimizing variance. We prove this by contradiction. Consider a sensor movement plan Z^S_{opt} that corresponds to a flow plan Z^V_{opt} determined by executing the minimum cost maximum weighted flow algorithm on G_V. Let this movement plan be non-optimal in terms of variance. This implies that there is a better movement plan, Z^S_x that can further minimize variance in the sensor network. By Corollary 1, a corresponding flow plan Z^V_x can be found in G_V. The amount of weighted flow in this plan is larger than the weighted flow achieved using plan Z^V_{opt}, which is a contradiction. Hence Z^S_{opt} is the optimal movement plan for sensors that minimizes variance.

We now prove that our *OMF* algorithm is optimal in terms of minimizing the number of sensor movement hops. We prove this by contradiction. Consider a sensor movement plan Z^S_{opt} that corresponds to a flow plan Z^V_{opt} determined by executing the minimum cost maximum weighted flow algorithm on G_V. Let this movement plan be non-optimal in terms of the number of sensor movement hops. This implies that there is a better plan, Z^S_x that can that can reduce at least one movement in the sensor network. By Corollary 1, a corresponding flow plan Z^V_x can be found in G_V. The number of movement hops (or overall cost) in this plan is less than the number of movement hops (or overall cost) achieved using Z^V_{opt}, which is a contradiction. Hence Z^S_{opt} is the optimal movement plan that minimizes the number of sensor movement hops. ■

We now discuss the time complexity of our algorithm. There are three phases in our algorithm while determining the optimal movement plan. The first is the construction of $G_V(V_V, E_V)$ and $G^m_V(\bar{V}, \bar{E})$, the second is determining the maximum flow in G^m_V, and the third is determining the minimum cost flow in G^m_V. The time complexity is dominated by determining the maximum flow and the minimum cost flow in G^m_V. Our implementations of the maximum flow algorithm is the Edmonds-Karp algorithm [11], and minimum cost flow algorithm is the one in [12]. The resulting time complexity using our implementations is $O(max\ (|\bar{V}||\bar{E}|^2, |\bar{V}|^2|\bar{E}|log|\bar{V}|))$. Here $|\bar{V}|$ and $|\bar{E}|$ denote the number of vertices and edges in G^m_V, and are given by, $|\bar{V}| = O(\bar{k}(\lceil \frac{Q}{R} \rceil^2))$, and $|\bar{E}| = O(\bar{k}H^2(\lceil \frac{Q}{R} \rceil^2))$, in which Q is the sensor network size and R is the region size.

F. Balancing Sensor Deployment and Movements

We now discuss how to extend the *OMF* algorithm to *optimally* solve two representative deployment problems that fall within the framework of our above general deployment problem.

a). Balancing Sensor Deployment: In some applications, the deployment goal is to ensure that the number of sensors among *all* regions is balanced [2]. In this problem, we have to minimize the global variance in the number of sensors among *all* regions from $\bar{k}_{ave}$, where $\bar{k}_{ave}$ is the average number of sensors per region. We now discuss how to modify G_V for this problem. The key issue here is how many sinks per region need to be created in G_V. Intuitively, it may appear that $\bar{k}_{ave}$ sinks per region need to be created. However, due to limited mobility, at final deployment, it may happen that some regions will have more than $\bar{k}_{ave}$ sensors. Creating $\bar{k}_{ave}$ sinks per region is not enough. On the other hand, the following Lemma shows that the final deployment plan that minimizes the global variance in the number of sensors among all regions from an arbitrary constant (k_{arb}), will minimize the global variance in the number of sensors among all regions from $\bar{k}_{ave}$

Lemma 2: Let k_{arb} be any arbitrary constant. The final deployment plan that minimizes the global variance in the number of sensors among all regions from k_{arb}, will minimize the global variance in the number of sensors among all regions from $\bar{k}_{ave}$.

Proof: At final deployment, let regions $r_1, r_2, r_3, \ldots, r_S$ contain $k_1, k_2, k_3, \ldots, k_S$ sensors respectively. The number of sensors $k_1, k_2, k_3, \ldots, k_S$, correspond to the final deployment that minimizes global variance in the number of sensors among all regions from k_{arb}. Clearly, the number of sensors in the network, $N = \sum_{i=1}^{S} k_i$. The final deployment variance from k_{arb} is given by $V_{arb} = \frac{1}{S}\sum_{i=1}^{S}(k_i - k_{arb})^2$. By our assumption, this is minimum. Therefore, $k_1^2 + k_2^2 + k_3^2 \ldots + k_S^2 - 2k_{arb}(k_1 + k_2 + k_3 \ldots + k_S) + Sk_{arb}^2$ is minimum. That is, $k_1^2 + k_2^2 + k_3^2 \ldots + k_S^2 - 2k_{arb}(N) + Sk_{arb}^2$. That is, $k_1^2 + k_2^2 + k_3^2 \ldots + k_S^2$ is minimum, since S, N and k_{arb} are constants. This implies that $k_1^2 + k_2^2 + k_3^2 \ldots + k_S^2 - 2\bar{k}_{ave}(k_1 + k_2 + k_3 \ldots + k_S) + S\bar{k}_{ave}^2$ is also minimum, since $\bar{k}_{ave}$ is a constant. That is, global variance in the number of sensors among all regions from $\bar{k}_{ave}$ is also minimum. ■

With the above Lemma, we create n_{max} sinks for each region in G_V, where n_{max} is the number of sensors in the region with the maximum number of sensors at initial deployment. All other construction rules for G_V remain the same. The following theorem shows the optimality of our solution to the problem of balancing sensor deployment.

Theorem 4: The deployment plan after executing the *OMF* algorithm with n_{max} sinks for each region in G_V will minimize the global variance in the number of sensors among all regions from $\bar{k}_{ave}$.

Proof: We first prove that executing the *OMF* algorithm with n_{max} sinks will minimize the variance in the number of sensors among all regions from n_{max}. We prove this by contradiction. Let Z_{opt} be the movement plan after executing the *OMF* algorithm that is optimum. optimum. Assume region i has more than n_{max} sensors after execution of *OMF* algorithm. That is, $k_i > n_{max}$. Since $n_i <= n_{max}$, at least one of the k_i sensors in region i comes from another region. Let this region be denoted by j ($i \neq j$). Let us denote the sensor moving from region i to region j as $\bar{s_1}$. If $k_j < n_{max}$, then we can construct a new movement plan, where sensor $\bar{s_1}$ stays in region j. Clearly this plan reduces the variance compared to Z_{opt}, which is a contradiction. Therefore $k_j \geq n_{max}$. Now, since region j initially had $\leq n_{max}$ sensors, and since it now has $\geq n_{max}$ sensors, and since a sensor has moved out of region j, this means that some other sensor $\bar{s_2}$ has moved from some region m to region j. Using the above argument, we can show that $k_m \geq n_{max}$. Therefore a sensor moves from some region p to region m.

From the above argument, we can see that there must be a path of the form $< i, j, m, p, \ldots >$, where each sensor has $\geq n_{max}$ sensors, and at least one sensor moves from region j to region i, region m to region j and so on. Since the number of regions is finite, there must exist a circular path in such a movement. Let us construct a new movement plan, where the sensors that moved across such a path do not move. Clearly, the variance remains the same, while movement hops is minimized in the new plan, which is a contradiction. Therefore, n_{max} sinks suffices for all sensors to move to some sink. By Theorem 1, the global variance from n_{max} is minimized in the corresponding plan (since we have n_{max} sinks).

Clearly, n_{max} is a constant. By Lemma 2, a deployment plan that minimizes variance in number of sensors from n_{max} among all regions, also minimizes the variance in number of sensors from $\bar{k}_{ave}$ among all regions. The theorem in hence proved.

■

b). Balancing Sensor Movements: In some cases, apart from the objectives of minimizing deployment variance and minimizing movement hops, it may be necessary to balance the remaining movement capability of the sensors in the network (especially if sensors are expected to move later). This is more true if the movement is a function of energy. Balancing the remaining mobility among sensor will translate to balancing remaining energy after final deployment. We propose a new cost assignment rule to G_V^m that ensures that apart from minimizing variance and movement hops at final deployment, the remaining movement distance among all sensors is also balanced. Between any two reachable regions i and j in G_V^m, the new cost of the edge from v_i^{in} to v_j^{out} is $d_{i,j} \times |E| \times H^2 + d_{i,j}^2$. The cost of edges from sink m in region i, vs_i^m to S_{sink}^v for all m is assigned as $-(2m-1) \times |E| \times (|E| \times H^3 + H^2)$. Other construction

rules for G_V^m remain unchanged.

Theorem 5: The movement plan after executing the *OMF* algorithm with the modified G_V^m will minimize the deployment variance, number of movement hops, and variance in terms of remaining movement distance among all sensors in the network.

Proof: Basically, the cost between reachable regions includes $\alpha \times d_{i,j} + d_{i,j}^2$. The large value of α will ensure that $\sum d_{i,j}$ and $\sum d_{i,j}^2$ are simultaneously minimized, which means overall movement distance, and the variance in movement distance among all sensors is minimized. The costs of edges from vs_i^j to S_{sink}^v is assigned such that weighted flow is still maximized, with the above cost assignment rule. The proof as such is very similar to the proof of Theorem 2. ■

V. Distributed Algorithms

In the above, we presented a centralized and optimal algorithm to our deployment using our weight-based methodology. We now present two algorithms to our problem that are distributed in their implementation using the weight-based methodology. Our first algorithm is called the Domain-based *OMF* (*D-OMF*) algorithm, and it extends from the *OMF* algorithm. Our second algorithm is called the Simple Pit-Peak based distributed (*SPP*) algorithm. While our distributed algorithms cannot produce optimal solutions in all circumstances, under certain deployment scenarios, solutions close to optimality can be produced as we discuss later. We first describe the *D-OMF* algorithm, followed by the *SPP* algorithm below.

A. *The Domain-based* OMF *algorithm*

In simple terms, the *D-OMF* algorithm is one, where the sensor network is divided into multiple domains, and each domain contains multiple regions. We let each domain obtain region information (number of sensors) *only* in their domain. The movement plan for variance minimization in each domain is independently determined with this information (without exchanging information with other domains) using the *OMF* algorithm. The Base-station can do this for each domain, or a special sensor in each domain can do so. The resulting time complexity of the algorithm for each domain is $O(max\ (|\bar{V}||\bar{E}|^2, |\bar{V}|^2|\bar{E}|log|\bar{V}|))$. Here $|\bar{V}|$ and $|\bar{E}|$ denote the number of vertices and edges in the virtual graph constructed for each domain, and are given by, $|\bar{V}| = O(\bar{k}(\lceil\frac{D}{R}\rceil^2))$, and $|\bar{E}| = O(\bar{k}H^2(\lceil\frac{D}{R}\rceil^2))$, in which D is the domain size and R is the region size. Note that the number of domains is $(\frac{Q}{D})^2$ for network size Q.

An illustration of the *D-OMF* algorithm is shown in Figures 4 (a) and (b). In Figure 4 (a), the sensor network has $8 \times 8 = 64$ regions. We divide the network into domains of size 4×4 (denoted as Domain size

$D = 4$). There are thus four domains, D_1, D_2, D_3, D_4 that are shaded and shown within dark borders in Figure 4 (b). Each domain will obtain their region information only, and the *OMF* algorithm we proposed above will be executed by each domain independently to determine the movement plan for variance minimization in the particular domain. This approach can only achieve local optima within each domain and cannot guarantee global optima in the sensor network. Nevertheless, it reduces the computational computational and messaging overhead, as the computations and messaging are done for each domain independently. In fact, when the Domain size is chosen carefully (based on initial deployment), solutions close to optimality can be obtained, as discussed below.

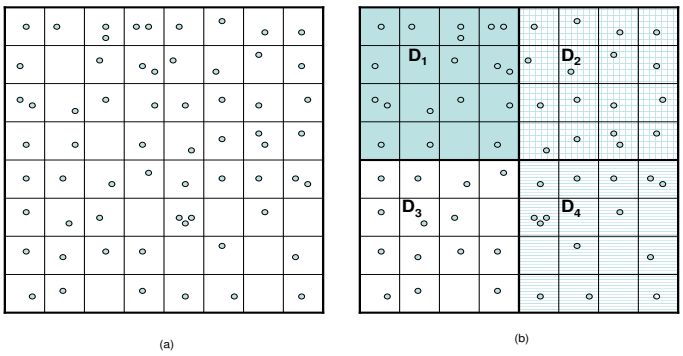

Fig. 4. The initial sensor network deployment (a) and after dividing the network into four domains (b)

In the following, we discuss the relationship between domain size D and the bias in the initial sensor deployment (denoted by σ). Let us for now assume that a large σ means more concentrated (biased) sensor deployment, while the deployment is more uniform for small σ. We highlight the issue of exploiting bias in the *D-OMF* algorithm with a specific type of deployment, called *group* deployment. In general, for large scale sensor networks deployment (especially in hostile/ in-accessible zones), it is likely that sensors are dropped from an airplane, targeted at chosen landmarks in the deployment field. That is, a *group* of sensors are launched in each landmark as the airplane flies over the landmarks. Such a deployment is called group deployment [13], [14]. Here, the ensuing deployment is biased (concentrated) in and around the landmarks. Intuitively speaking, if the deployment is optimized in each group independently (without global synchronization), then the resulting deployment will not be very different from the optimal case. Hence, we can see that the domain size is closely related to the group size. The important issue here is how to choose optimal domain size D. Deriving an analytical expression for optimal D, group size and σ for a general deployment scenario in the presence of limited mobility sensors is very difficult, if not impossible. We study this issue using extensive simulations in the next section.

B. The Simple Peak-Pit based Distributed algorithm

In the above, we proposed a distributed way to execute the *OMF* algorithm among multiple domains in the sensor network. We now propose the Simple Pit-Peak (*SPP*) based algorithm to our deployment problem, that is local, light-weight and *purely* distributed. In the *SPP* algorithm, regions request sensors from adjacent regions, with weights attached to each request. As before, requests with larger weights are given higher priority when compared to requests with smaller weights, while simultaneously preferring shorter movement hops to satisfy requests. We first discuss some important notations used in the algorithm description. Regions in the network are classified into three types: $pits$, $peaks$ and $forwarders$. A pit is a region whose number of sensors is less than $\bar{k}$ and not more than any of its neighboring regions. A $peak$ is a region whose number of sensors is larger than any of its neighboring regions. All other regions are $forwarders$. We define an *over-*$\bar{k}$ $forwarder$ as a $forwarder$ with more than $\bar{k}$ sensors, and denote the $richest$ neighbor of a region as a neighbor with the largest number of sensors.

In the *SPP* algorithm, a pit i will request $\bar{k} - n_i$ sensors in its request (REQ). The pit i will assign different weights to each of the $\bar{k}-n_i$ requested sensors as, $w_i^j = 1, 3, 5, \ldots, 2j-1$, where $1 \leq j \leq (\bar{k}-n_i)$ (as before). Here, we let only $pits$ send $REQs$, so that *non-pit* regions will not compete with $pits$ during requests to ensure that more deficient regions will be given priority. A REQ generated will be forwarded towards progressively richest neighbors to increase likelihood of $REQs$ arriving at *over-*$\bar{k}$ $forwarders$ or $peaks$ on shorter paths. Recipients receiving $REQs$ will sort all the requested sensors in the $REQs$ by $weights$ and serve those with larger $weights$ first. Ties are broken by fulfilling requests with shorter paths first. We call the neighbors chosen for the next hop as $tried$ neighbors.

Algorithm 3 shows the pseudocode of our *SPP* algorithm. It is executed by each region i independently and is driven by several events. Using inter-region communications, a region leader will be elected for coordination. Each region leader obtains the number of sensors in its region, and its four adjacent neighboring regions. The region leader of each *pit* will send an REQ to its *richest* neighbor, requesting the number of sensors needed. If multiple *richest* neighbors exist, ties are broken randomly. If some regions have no sensors, they can be assisted by neighboring region leaders in sending our requests. We discuss this issue in further detail later.

Due to limited mobility, when requests are sent out, it is important that path feasibility should be maintained during the selection of next hop $forwarder$. This means there should exist at least one mobile sensor on any continuous H hop segment of the path a REQ traverses. Otherwise, mobile sensors

Algorithm 3 Pseudocode of the *SPP* algorithm run by region i

```
1:  Region leader selection
2:  while TRUE do
3:    switch type of event
4:    case region i becomes a pit:
5:      send REQ to richest untried neighbor;
6:    case receive REQ:
7:      put REQ into Queue(i);
8:      if region i is an over-k̄ forwarder, then
9:        select REQs in Queue(i) to serve by
            weights and path lengths;
10:       send ACKs, move sensors and/or forward
            REQs accordingly;
11:     else if region i is a peak, then
12:       select REQs in Queue(i) to serve by
            weights and path lengths;
13:       send ACKs, move sensors and/or send
            FAILs accordingly;
14:     else
15:       forward REQ to richest neighbor;
16:   case receive ACK for pit j:
17:     forward ACK to j if i ≠ j;
18:   case receive FAIL for pit j:
19:     resend REQ to richest untried neighbor;
20:   case detect hole neighboring region j:
21:     if region i can provide sensor, then
22:       move a sensor to j after random delay;
23:   end switch
24: end while
```

on the other side of the segment will not be able to move back to the requesting pit due to limited mobility. In case there is not enough mobile sensors in a certain segment with H hops on the path, the requested number of sensors in the REQ message should be adjusted since we can never move enough sensors back on the path. All the intermediate $forwarders$ will reserve enough number of mobile sensors to guarantee the feasibility of the path.

When $REQs$ are forwarded to *over-*$\bar{k}$ $forwarders$ or $peaks$, some of them may or may not get served. Considering that the REQ with largest $weight$ requested sensor may not always come first, the *over-*$\bar{k}$ $forwarder$ or $peak$ will put the $REQs$ into its queue and serve them in periodic intervals of time. When serving multiple requested sensors with the same $weights$, those with shorter paths will be served first. An *over-*$\bar{k}$ $forwarder$ will send $ACKs$ back to the $pits$ whose $REQs$ contains sensors that will be served, and forward the $REQs$ if not all sensors can be served. Those forwarded $REQs$ will be updated if part of the requested sensors are served eventually. A $peak$ will send $ACKs$ back to the $pits$ whose $REQs$ contains sensors that will be served, and send $FAILs$ back to $pits$ if not all requests can be served [2]. Sensors will start moving after $ACKs$ are sent, following the reserved paths of the corresponding $REQs$. After receiving the $ACK(s)$ and mobile sensor(s), each pit will inform its neighbors its new sensor number, and $REQs$ are generated if need be. After a pit or $forwarder$ receives a $FAIL$, it will release the reserved path and resend $REQs$ to its $richest\ untried$ neighbor and so on. The algorithm stops when each pit has either obtained $\bar{k}$ sensors or a certain number of $REQs$ have been sent out.

[2]We do not let recipients of requests choose shortest return paths as such paths may be *blocked* due to limited mobility sensors.

It may happen some regions have no sensors in them (denoted as *holes*) after initial deployment. A hole can be filled by one of its neighbors with extra mobile sensors, after coordination with other non-empty neighbors. In case a hole cannot be filled directly due to none of its neighbors being able to provide an extra sensor, one of its neighbors can become its proxy region via the same mechanism discussed above. In the extreme case when all of a hole's neighbors are empty, the hole may be filled by sensors, or have a proxy region leader later when some of its neighbors obtain sensors during the execution of the *SPP* algorithm.

VI. Performance Analysis

In this section, we report our experimental data to study the performance of the *OMF* algorithm, the *D-OMF* algorithm, and the *SPP* algorithm. We also study sensitivity of performance of our algorithms to various sensor and network parameters.

A. Performance Metrics and Evaluation Environment

1) Performance Metrics: We have three major performance metrics in this paper. The first is the *Variance Improvement* (denoted by *VI*) at final deployment after sensors have finished movements. It is defined as $VI = (\frac{Var_{in} - Var_{out}}{Var_{in}}) \times 100$, where, Var_{in} is the variance at initial deployment and Var_{out} is the variance at final deployment. Our second metric is the number of sensor *Movement Hops* per percent variance improvement (denoted by *MH*). It is defined as $MH = \frac{M}{VI}$, where M denotes the total number of sensor movement hops. The reason we define MH as a ratio is because, it is more fair to compare number of hops per improvement in variance, than just the number of hops.

Our third metric is the messaging overhead incurred by our algorithms, which is defined as the *Packet Number* per region (denoted by *PN*). Denoting P as the total number of packets (or messages) sent, and denoting S as the number of regions, we have $PN = \frac{P}{S}$. Physically speaking, VI captures the improvement in deployment as a result of our algorithms, while MH and PN reflect the overhead in terms of sensor movement hops and messaging overhead. The packet number for our *OMF* algorithm is calculated based on a simple protocol. After initial deployment, an elected region-head in each region sends a packet to Base-station (located in the center of the network) with information on the number of sensors in its region. The packets are forwarded along shortest paths through other regions towards the Base-station. After the Base-station receives all packets and determines a movement plan, it sends one packet to each region in the reverse path, informing regions of its movement plan. A similar protocol is

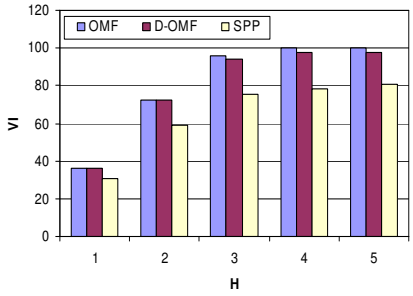

Fig. 5. Sensitivity of VI to H

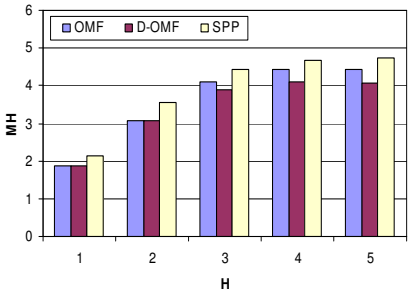

Fig. 6. Sensitivity of MH to H

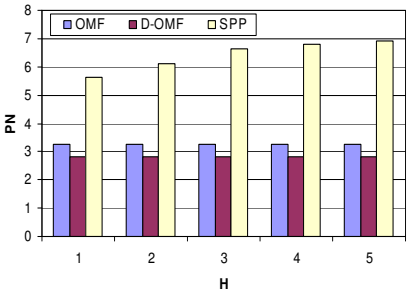

Fig. 7. Sensitivity of PN to H

assumed for the *D-OMF* algorithm, where the regions in each domain will forward packets to a special sensor in the domain, which executes the algorithm and forwards a movement plan to each region in the domain. Note that, there can be other versions of the above protocols, like direct relaying of messages, row-wise (or column wise) message delivery etc.

2) Evaluation Environment: We denote the number of regions in the network as $n \times n$ (represented in the figures as simply n). Our default value is 8×8. The default desired number of sensors per region is $\bar{k} = 3$ and maximum number of hops a sensor can move is $H = 3$. By default, the number of sensors initially deployed is $n \times n \times \bar{k}$. For the *D-OMF* algorithm, we choose the domain size D as $D = \frac{n}{2}$. In the *SPP* algorithm, a $peak$ and *over-*$\bar{k}$ *forwarder* will batch up the coming $REQs$ in a time period to serve. In our simulation, the time period is given by $t_u \times n$, in which t_u is the message transmission delay between two neighboring regions.

We conduct our simulations on a custom simulator. For initial deployment, our simulator uses a topology generator for 2D-Normal distribution. We use σ to denote the degree of concentration of initial sensor deployment in the center of the deployment field. Larger values of σ implies more concentrated deployment in the center of the field. When $\sigma = 0$, the deployment is uniform. The default value is $\sigma = 4$. All data reported here were collected across 10 iterations, and averaged. Our implementations of the maximum flow algorithm is the Edmonds-Karp algorithm [11], and minimum cost flow algorithm is the one in [12].

B. Performance Results

1) Performance comparison of the OMF, D-OMF and SPP algorithms: We first study the sensitivity of VI, MH and PN to mobility capacity H for *OMF*, *D-OMF* and *SPP* algorithms, when sensors are initially deployed once, targeted at the center of the sensor network. The configuration is our default one. From Figure 5, we observe that the *OMF* algorithm (being optimal) performs best in terms of VI. We observe that the *D-OMF* algorithm performs quite close to the *OMF* algorithm in all cases, while the

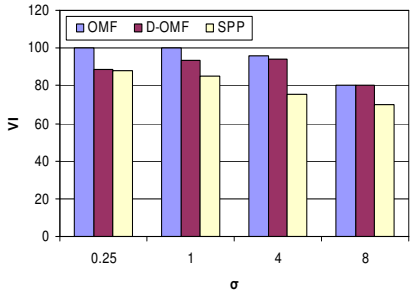

Fig. 8. Sensitivity of VI to σ

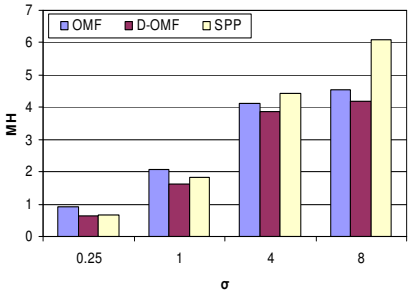

Fig. 9. Sensitivity of MH to σ

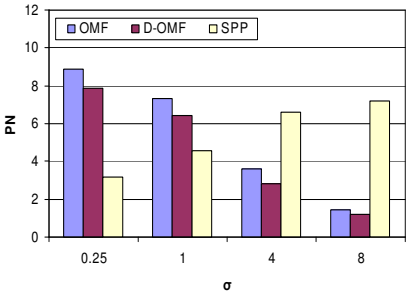

Fig. 10. Sensitivity of PN to σ

performance of the *SPP* algorithm is quite good for smaller values of H. Figure 5 also shows the effects of limited mobility on VI. When H increases, VI increases in all algorithms as increased movement ability in general helps move sensors to needy regions farther away. However, the increasing trend is only upto a point. When $H \geq 4$, *OMF* can achieve the maximum improvement of 100% VI, while the VI in *D-OMF* and *SPP* algorithms shows a constant trend with increasing H. For the *OMF* algorithm the upper bound of 100% is reached, because enough movement choices can be optimally determined by the *OMF* algorithm with large H. For the *D-OMF* algorithm, since we execute our algorithm independently in each domain, some mobility choices will be prevented from being exploited unlike in the case of the global *OMF* algorithm. Hence VI cannot be 100%, and VI will reach an upper bound (below 100%) for the *D-OMF* algorithm. In the *SPP* algorithm, the upper bound on VI below 100% is due to non-optimal requests and responses during sensor movements in the algorithm.

Figure 6 shows that for small H, MH also increases when H increases. This is because more movement hops are incurred with more H, consequently increasing MH. However, we also observe that MH shows a constant trend for large values of H in all our algorithms. This is because of the upper bound in VI discussed earlier. Since VI does not increase beyond a certain value, the total number of movements M (for a particular algorithm) is a constant. Since $MH = \frac{M}{VI}$, there is thus a constant trend of MH for all algorithms for large H. When comparing the MH of the algorithms, we see that the *D-OMF* algorithm has less MH than that of the *OMF* algorithm. This is because in the *D-OMF* algorithm, we execute the algorithm independently in each domain. Since the algorithm only optimizes VI in each domain, the possibility of long movement paths (between domains) is avoided here. However, the *OMF* algorithm will exploit *all* available paths throughout the network to optimize VI. Consequently, there will be paths in the *OMF* algorithm that are long, which makes the total number of movement hops M in the *OMF* algorithm much larger than that of the *D-OMF* algorithm, resulting in the MH for the *D-OMF* algorithm being less than that of the *OMF* algorithm. For the *SPP* algorithm the high value of MH is due to the

lower VI that the *SPP* algorithm achieves compared to the other two algorithms. In Figure 7, we can see that PN in the *OMF* and *D-OMF* algorithms are constant, since packet number does not depend on H for these algorithms. PN in the *SPP* algorithm is larger than that of the *OMF* and *D-OMF* algorithms due to many local requests and responses. PN increases with H in the *SPP* algorithm, because the $REQs$ can be forwarded further due to the feasibility of longer paths when H increases.

Figures 8, 9 and 10 show the sensitivity of our performance metrics to σ for our algorithms. In Figure 8, VI decreases when σ increases for the *OMF* and *SPP* algorithms. Since larger σ implies more concentrated initial deployment, it is harder for regions near the boundary to find sensors under mobility constraints, which decreases VI for the *OMF* and *SPP* algorithms. We also see that VI of the *D-OMF* and *SPP* algorithm become closer to that of the *OMF* algorithm as σ increases. This is because, the amount of mobility choices that the *OMF* algorithm can exploit is not significantly more than that of the other algorithms, when deployment is highly concentrated.

We wish to emphasize on the interesting trend of VI for the *D-OMF* algorithm. Note that VI is not monotonically decreasing with σ (unlike other algorithms). Recall from Section V-A that the feature of the *D-OMF* algorithm is how to effectively exploit bias (σ). Figure 8 demonstrates this feature. From Figure 8, the important trend to observe is the *difference* in VI between the *OMF* and *D-OMF* algorithms. The difference becomes smaller as σ increases, and both algorithms have same VI when $\sigma = 8$. The non-monotonic decrease in VI is due to this difference (which monotonically decreases). In our experiments here, we set domain size $D = 4$ for the default 8×8 network. As we can see, this is not the best choice under all σ, since for small σ, the difference in VI is quite large. We study the issue of the sensitivity of domain size D and σ in further detail in the next sub-section.

In Figure 9, we see that MH increases as σ increases for our algorithms. This is mainly because of the reduction in VI with increasing σ. Note here that MH is lower for the *SPP* algorithm when σ is small. This is because, when deployment is more uniform (smaller σ), more $pits$ can find enough *over-*$\bar{k}$ $forwarders$ or $peaks$ nearby, which causes a reduction in overall sensor movements. In Figure 10, we see that the PN for the *OMF* and *D-OMF* algorithms decreases with σ, since the number of sensors farther away from the center of the network decreases with increased σ. On the other hand, PN increases with σ in the case of the *SPP* algorithm, since the increase in σ means that the bias increases, resulting in more requests and responses. We also observe that when σ is less, the PN in the *SPP* algorithm is lower than that of the *OMF* and *D-OMF* algorithms. This is because, more *pits* can find enough $over - \bar{k} forwarders$ or *peaks* in the *SPP* algorithm when the deployment is more uniform, further highlighting the fact that

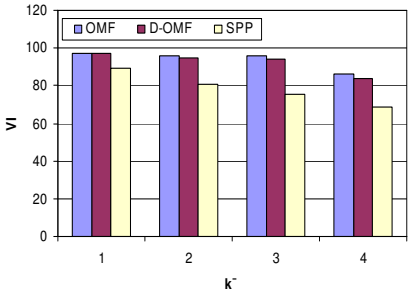

Fig. 11. Sensitivity of VI to $\bar{k}$

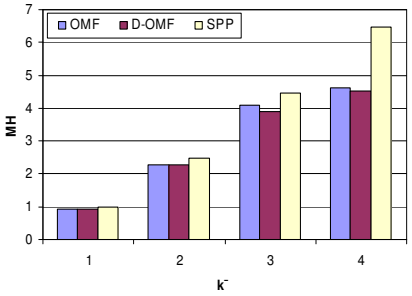

Fig. 12. Sensitivity of MH to $\bar{k}$

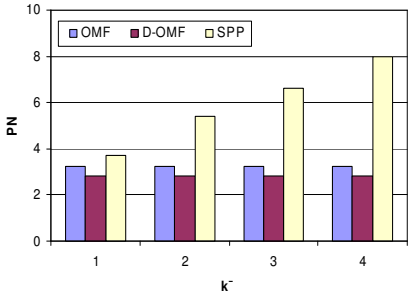

Fig. 13. Sensitivity of PN to $\bar{k}$

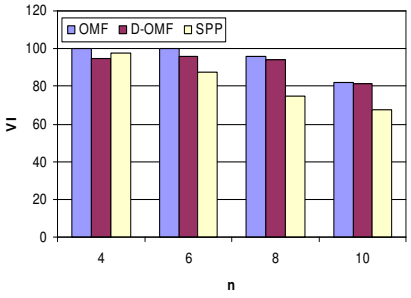

Fig. 14. Sensitivity of VI to n

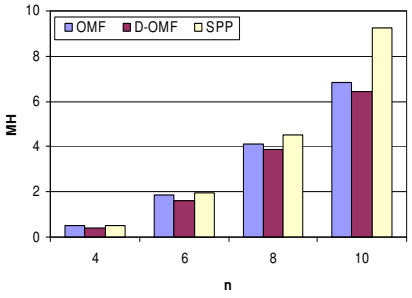

Fig. 15. Sensitivity of MH to n

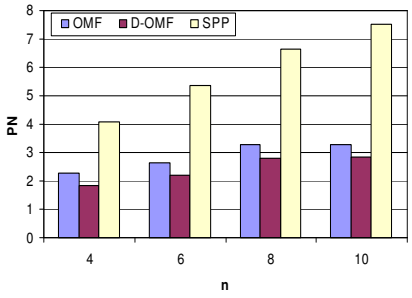

Fig. 16. Sensitivity of PN to n

the distributed *SPP* algorithm achieves less overhead under favorable deployment conditions.

We now study the sensitivity of our performance metrics to $\bar{k}$ for our algorithms. In order to compare the sensitivity to $\bar{k}$ fairly, the number of sensors initially deployed is fixed as $8 \times 8 \times 3 = 192$ for all cases (other parameters are default values). From Figures 11 and 12, we can see that an increase in $\bar{k}$ causes a decrease in VI and an increase in MH in our algorithms. When $\bar{k}$ increases, the objective becomes harder, which causes this trend. We can also see that the *D-OMF* algorithm performs quite close to the *OMF* algorithm in all cases. An interesting observation here is that, when $\bar{k}$ is small, the performance of the *SPP* algorithm in all metrics compares quite favorably with the other algorithms. This is because, when the deployment objective is relatively mild (less $\bar{k}$), local requests and responses suffices for good performance. Once again, the PN for the *OMF* and *D-OMF* algorithms in Figure 13 is independent of $\bar{k}$ and hence is constant. The PN of the *SPP* algorithm is similar to the other algorithms for less $\bar{k}$, and increases with increasing $\bar{k}$ since more requests and responses are generated when $\bar{k}$ increases.

In Figures 14, 15 and 16, we can see that as n increases, VI decreases and both MH and PN increase for our algorithms. A larger n implies a larger network, which makes more regions near the boundary of the network unable to get sensors, and thus VI decreases. Also, sensors need to travel longer distances, which increases MH and PN.

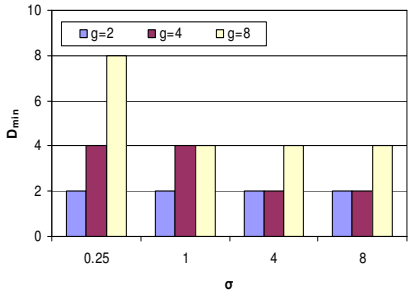

Fig. 17. Sensitivity of D to σ under different g

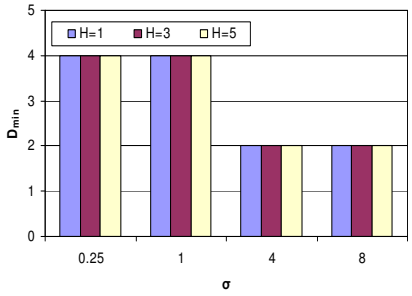

Fig. 18. Sensitivity of D to σ under different H

2) Sensitivity of D to σ, g and H in the D-OMF *algorithm:* Here, we report our observations on the sensitivity of the domain size D (in the *D-OMF* algorithm) to σ under group deployment with different group sizes g and different H in our default 8×8 network with $\bar{k} = 3$. Recall that, the group deployment is one, where the sensor network (containing $n \times n$ regions) is divided into many groups, and sensors are deployed at the center of each group. Each group in the sensor network contains $g \times g$ regions (denoted as group size g). For example, in our 8×8 network; when group size $g = 2$, there are 16 groups; when $g = 4$, there are four groups; and when $g = 8$, there is only one group. In all simulations, the term σ once again denotes degree of concentration at the center of each group during initial deployment.

Our goal here is to study the sensitivity of performance (in terms of *Variance Improvement* VI) of the *D-OMF* algorithm to initial deployment bias. Specifically, we want to divide the network into multiple domains and study the sensitivity of domain size D to σ, g and H in order to obtain optimum values of the domain size D for acceptable performance. Towards this extent, we define a new metric, namely the Minimum domain size (D_{min}). D_{min} for the *D-OMF* algorithm is the minimum domain size D such that, error from optimality in VI at final deployment can be tolerated. The Error Tolerance (ET) is defined as, $ET = \frac{VI_{op} - VI_D}{VI_{op}}$, where VI_{op} is the optimal VI and VI_D is the VI obtained with domain size D in the *D-OMF* algorithm. The minimum domain size D_{min} is the smallest domain size D such that Error Tolerance $ET \leq \epsilon$ for an appropriately chosen ϵ that is decided by the application. In our simulations we set $\epsilon = 0.1$. Physically, this means that the domain size D_{min} is the smallest domain size such that error in VI from optimal case is $\leq 10\%$.

In Figure 17, we see the sensitivity of D_{min} to σ under different g, when H is fixed as 3. First, we see that in all instances a domain size equal to the group size performs acceptably well (i.e., $ET \leq 10\%$). For example, in Figure 17, $D_{min} = 2$, $D_{min} = 4$ and $D_{min} = 8$ are the minimum domain sizes needed for acceptable performance for groups sizes 2, 4 and 8 respectively when $\sigma = 0.25$. This validates our earlier claim made in this realm in Section V-A that smaller domain sizes will suffice if we can effectively

exploit bias deployment. Interestingly, we observe that as σ increases, a domain size D_{min} smaller than the group size (specifically, $D_{min} = \frac{g}{2}$) also performs acceptably well. For example, for larger σ, $D_{min} = 4$ performs acceptably well when $g = 8$, and $D_{min} = 2$ performs acceptably well when $g = 4$ in Figure 17. The reason behind this trend is the following. When σ is large, more sensors are likely to be concentrated in the center of the group at initial deployment. In this case, it is very likely that sensors are uniformly balanced in all directions surrounding the center of each group, in which case $D_{min} = \frac{g}{2}$ will suffice to acceptably exploit this bias. When σ is very small, the initial deployment is scattered over a larger area. Dividing the area into domains when the deployment is scattered over a large area will not acceptably exploit the bias (when compared to global *OMF* algorithm). Hence, D_{min} decreases as σ increases. Recall from our discussions earlier on the trend of Figure 8 for the *D-OMF* algorithm (where the domain size was set as $D = 4$). When σ was small, the difference in VI between the *D-OMF* and *OMF* algorithm was quite large when $D = 4$ in Figure 8. However, when σ increases, the difference became quite less as the same domain size ($D = 4$) can better exploit the bias with increasing σ, as explained above.

In Figure 18, we study the sensitivity of the domain size D_{min} to σ under different H, when g is fixed as 4. Here again, we see that domain size equal to group size ($D_{min} = g = 4$) performs acceptably well for smaller σ. We also see that $D_{min} = 2$ performs acceptably well for $g = 4$ when σ increases. It is interesting to note that D_{min} is not sensitive to H for fixed σ and g. This is because, the performance of the *D-OMF* algorithm depends on how to exploit deployment bias such that solutions for local optimum are acceptable. When σ and g are fixed, the deployment bias is not sensitive to H. Consequently, D_{min} is also not sensitive to H when σ and g are fixed.

VII. Discussions

In the following, we provide discussions on two issues: a) combinedly executing our algorithms and b) some implementation issues in our algorithms.

a). Combined execution of our algorithms: Although each of our algorithms can execute independently, they can also be combined in practice. We first discuss how to combine the *OMF* and *D-OMF* algorithms. Consider a situation when sensors are over deployed. In case, we want to execute the *D-OMF* algorithm, it is reasonable to first balance the number of sensors in each domain to be fair to all domains. The *OMF* algorithm can be executed first in the whole network under the consideration of each domain as one region, with the objective of *optimally* balancing the number of sensors in each domain. After this, the *D-OMF* algorithm is executed in each domain independently. If sensors die eventually and have to be

redeployed, the above process can repeat over time.

Similarly, the *OMF* and *SPP* algorithm can be combined as follows. After optimizing deployment with the *OMF* algorithm, as time goes on, some sensors may die due to faults, energy losses etc. To replace such sensors, we do not have to re-execute the *OMF* algorithm. The light-weight *SPP* algorithm suffices here to request closer sensors. Over longer periods of time, when many sensors die (or under sensor re-deployment), the *OMF* algorithm can be executed. The principle of the above approach is to periodically optimize deployments in longer intervals of time, while between intervals, local requests and responses will be taken care of by the *SPP* algorithm.

An approach to combine the *D-OMF* and *SPP* algorithms is the following. The key shortcoming of the *D-OMF* algorithm is that, there may not enough sensors per domain (e.g., the *OMF* algorithm cannot balance number of sensors per domain). In such cases, the light-weight *SPP* algorithm can be executed near the borders of domains to locally request sensors from other neighboring domains. Once enough sensors are obtained in each domain, the performance of the *D-OMF* algorithm can be dramatically improved.

b). Arbitrary sensor movement directions and Network partitions: In Section II, we assumed that sensors can move only to regions in its adjacent left, right, top and bottom directions only. We now discuss how to extend to the case where sensors can move to regions in any arbitrary direction. For our *OMF* and *D-OMF* algorithms, only the construction of the virtual graph G_V changes. In G_V, we now have to add new edges (with corresponding costs and capacities) from a region to all newly *reachable* regions corresponding to arbitrary directions of sensor movements. In the *SPP* algorithm, there are now more neighbor choices to forward a request, and extra feasible paths can be reserved while sensor move back to satisfy requests.

In Section II, we assumed that the sensor network is not partitioned. In some situations it is likely that the network is partitioned, whereby, sensors in one part of the network may not be able to communicate with sensors in another part. In such cases, we have to repair such partitions, while still being constrained by mobility distance. In the approach proposed by Wu and Wang [2], empty holes are filled by placing a *seed* from a non-empty region to a hole. We can apply the algorithms in [2] to repair partitions in our case. However, we are still constrained by the mobility in sensors. Addressing the issue of repairing network partitions optimally using limited mobile sensors is a part of our on-going work.

c). Failures in Sensor Networks: In some situations sensors can fail or become faulty. For example, a sensor may be physically damaged preventing it from moving, or the sensors could make erroneous movements. Recall that in our algorithms, when sensors are over-deployed, the extra sensors will not move. Such extra sensors can help to correct sensors failures and faults eventually. This can be done

by periodic execution of our algorithms or suitably combining their execution (as discussed above). In some cases, there can also be communication failures among sensors in the network. Examples could be a packet from a sensor not reaching the base-station in the *OMF* algorithm, or requests/ responses getting lost in the *SPP* algorithm. A rigorous study of limited mobility sensor networks deployment under failures will be part of our future work.

VIII. Related Work

In this paper, we addressed a sensor networks deployment problem using limited mobility sensors. In this section, we discuss some important related work in the areas of general sensor networks deployment and works where sensor mobilities are used for deployment. In the simplest case, sensors are deployed randomly [15], [16]. Extending random deployment further, some recent work like [13], [14] have appeared, where sensors are randomly deployed in *groups* in the deployment field. The distribution pattern of group deployment is exploited for localization purposes. Another type of deployments follows incremental strategies, where sensors are deployed iteratively after making some measurements on the quality of previous partial deployments [17], [18], [19], [20]. However, the key shortcoming is that such approaches need to be conducted by an external mobile robot or a human being and as such are restrictive, especially in large-scale or hostile zone deployments.

More recently, mobility is sensors has been leveraged for deployment [1], [2], [3] and [4]. In works like [1], [3] and [4], the goal is uniform coverage of the network. The coverage here means that every point in the network is covered by at least *one* sensor. The approach used in [1], [3] and [4] is balancing of sensor virtual forces for uniform deployment. Two sensors may repel or attract each other based on their distance. At each iteration, sensors move to achieve a better force balance, and sensors stop moving, when a force equilibrium is reached. However, the virtual force approach introduces unnecessary movements during force balance, which cannot be tolerated by limited mobility sensors. Another difference is that all of the above works focus on one-coverage of the sensor network, while we are addressing a general variance minimization problem. Another mobility assisted deployment work is [2], where the objective is load balancing sensor deployment. In [2], the sensor network is initially divided into 2-D clusters. The problem is to ensure that starting from an initial deployment, the number of sensors in all clusters in the sensor network be the same. Based on efficiently scanning the clusters in two stages (row-wise and column-wise), sensors determine to which cluster they have to move. One drawback of [2] is that the ratio of number of hops in their algorithm and the optimal case is bounded by a factor of 2. Limited

mobility sensors cannot tolerate so many unwanted moves. Also, the problem in [2] is just a special case of a general deployment problem addressed in our paper.

There has also been some other work in utilizing sensor movements for desired deployment. In [21], algorithms are proposed in order to let sensors relocate to new positions in a 2-D dimensional grid based sensor network. The relocation process can be triggered by events, or when existing sensors in other points in the network become non-functional. However, such relocation is done such that it should not compromise existing functionality of the network. In [22], a set of algorithms are designed that enable sensor to move to events in the deployment field. The algorithms are designed to be less energy consuming and computationally mild for the sensors. In [23], the deployment problem is for sensors to maintain a regular hexagonal structure. In [23], sensors periodically detect errors in network invariants, and move to correct the violations of invariants. More recently, a new paradigm of mobility called parasitic mobility has been proposed in [24] and [25]. In this paradigm, sensors themselves are static, but utilize the mobility of external agents to relocate themselves. The sensors here selectively *engage* and *disengage* from external mobile agent(s) (that move in the network) whenever the sensors need to relocate themselves, to attain desired deployment.

We summarize the differences of our work from the above mobility assisted deployment work here. The first major distinguishing feature of our work from all the above works is that we focus on limited mobility sensors. In all of the above works sensor mobility is unlimited. Under limited mobility sensors, we are fundamentally constrained by the movement choices available to us (compared to unlimited mobility) during deployment. The consequence is increased importance that needs to be accorded during each sensor movement. Given this, the fact that our problem is to optimize both the final deployment and sensor movements makes our problem quite different and more challenging. Second, our problem is different from the above, in that we minimize the *variance* among the regions in the network. In this paper, we proposed a novel *weight-based* methodology, and an optimal centralized algorithm and distributed algorithms following from the weight-based methodology for our deployment problem.

We have done some preliminary work in limited mobility deployment in [26]. There, we addressed a problem of maximizing number of regions in the network with at least one sensor, with limited mobility sensors. The problem we address in this paper is minimizing variance, which is a non-linear objective problem. Another difference is that, the sensors were capable of only moving only *once* to a *fixed* distance in [26]. Here, we have a more general mobility model, where only the maximum sensor movement distance is limited. This paper proposes a distributed algorithm while our work in [26] did not.

IX. Final Remarks

In this paper, we defined a general sensor network deployment problem under limited mobility sensors and proposed three algorithms to solve the problem. Our methodology was to transfer the non-linear variance/movement minimization problem into a linear optimization problem through appropriate weight assignments to the regions. During sensor movements across the regions, larger weight regions are given higher priority compared to smaller weight regions, while simultaneously minimizing cost of movements. Based on the above methodology, we proposed three algorithms, namely the *OMF* algorithm, the *D-OMF* and the *SPP* algorithm for our problem. Using analysis and extensive simulations, we demonstrated the performance of our algorithms.

Our on-going work addresses the issue of repairing network partitions with limited mobility sensors. Secondly, we plan to study the issues of limited mobility sensors in other applications. As part of our future work in this realm, we are investigating the applications of limited mobility sensors in sensor tracking systems. The challenge is how to design algorithms that can exploit the limited mobility in sensors, and algorithms for provisioning sensors in the network such that tracking efficiency throughout the network and some specific hot spots in the network can be improved.

References

[1] G. Wang, G. Cao, and T. La Porta, "Movement-assisted sensor deployment," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Hong Kong, March 2004.

[2] J. Wu and S. Wang, "Smart: A scan-based movement-assisted deployment method in wireless sensor networks," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Miami, March 2005.

[3] A. Howard, M. J. Mataric, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Proceedings of International Symposium on Distributed Autonomous Robotics Systems (DARS)*, Fukupka, Japan, June 2002.

[4] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, San francisco, April 2003.

[5] D. Lymberopoulos and A. Savvides, "Xyz: A motion-enabled, power aware sensor node platform for distributed sensor network applications," in *Proceedings of International Symposium on Information Processing in Sensor Networks (IPSN)*, April, Los Angeles 2005.

[6] "http://www.darpa.mil/ato/programs/shm/index.html," .

[7] A. Howard, M. J. Mataric, and G. S. Sukhatme, "Relaxation on a mesh: a formation for generalized localization," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Maui, November 2001.

[8] B. Karp and H.T. Kung, "Greedy perimeter stateless routing for wireless networks," in *Proceedings of ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Boston, August 2000.

[9] F. Ye, A. Chen, S. Lu, and L. Zhang, "A scalable solution to minimum cost forwarding in large sensor networks," in *Proceedings of International Conference on Computer Communications and Networks (ICCCN)*, Arizona, October 2001.

[10] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proceedings of ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Rome, July 2001.

[11] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, "Introduction to algorithms," in *MIT Press*, 2001.

[12] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," in *J. Algorithms 22*, 1997.

[13] W. Du, L. Fang, and P. Ning, "Lad: Localization anomaly detection for wireless sensor networks,," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rhodes Island, April 2005.

[14] Y. Zou and K. Chakrabarty, "Uncertainty-aware sensor deployment algorithms for surveillance applications," in *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, San Francisco, CA, December 2003.

[15] S. Shakkottai, R. Srikant, , and N. B. Shroff, "Unreliable sensor grids: Coverage, connectivity and diameter," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, San francisco, April 2003.

[16] H. Zhang and J. C. Hou, "Maintaining coverage and connectivity in large sensor networks," in *The Wireless Ad Hoc and Sensor Networks: An International Journal*, March 2005.

[17] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan, and K. Saluja, "Sensor deployment strategy for target detection," in *Proceedings of ACM international conference on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, September 2002.

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

[19] V. Isler, K. Daniilidis, and S. Kannan, "Sampling based sensor-network deployment," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sendai, Japan, September 2004.

[20] N. Bulusu, J. Heidemann, and D. Estrin, "Adaptive beacon placement," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, Phoenix, AZ, April 2001.

[21] G. Wang, G. Cao, T. La Porta, and W. Zhang, "Sensor relocation in mobile networks," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Miami, March 2005.

[22] Z. Butler and D. Rus, "Event-based motion control for mobile sensor networks," in *IEEE Pervasive Computing, 2(4), 34-43*, October-December 2003.

[23] H. Zhang and A. Arora, "Gs3: Scalable self-configuration and self-healing in wireless sensor networks," in *Computer Networks (Elsevier), 43(4): 459-480*, November 2003.

[24] M. Laibowitz and J.A. Paradiso, "Parasitic mobility for pervasive sensor networks," in *ACM Workshop on Applications of Mobile Embedded Systems (WAMES)*, Boston, June 2004.

[25] M. Laibowitz and J.A. Paradiso, "Parasitic mobility in dynamically distributed sensor networks," in *International Conference on Pervasive Computing (PERVASIVE)*, Munich, May 2005.

[26] S. Chellappan, X. Bai, B. Ma, and D. Xuan, "Sensor networks deployment using flip-based sensors," in *Proceedings of IEEE International Conference on Mobile AdHoc and Sensor Systems (MASS)*, Washington D.C., November 2005.