Deploying Wireless Sensor Networks under Limited Mobility Constraints

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

Abstract

In this paper, we study the issue of sensor networks deployment using limited mobility sensors. By limited mobility, we mean that the maximum distance that sensors are capable of moving to is limited. Given an initial deployment of limited mobility sensors in a field 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 regions. In this methodology, the regions are assigned weights corresponding to the number of sensors needed. During sensor movements across 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 the Optimal Maximum Flow based (*OMF*) centralized algorithm. Here, the optimal movement plan for sensors is obtained based on determining the minimum cost maximum weighted flow to the regions in the network. We then propose the Simple Peak-Pit 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.

This work was partially supported by NSF under grants No. ACI-0329155 and CAREER Award CCF-0546668. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

I. INTRODUCTION

Provisioning sensors with mobility is a topic that has received significant attention recently. The XYZ sensor platform in [1], sensors in DARPA's self-healing minefield program in [2], the Robomote platform in [3] are recent instances of mobility enabled sensor models. Basically, the motivation behind this line of activity is the significant advantages that today's sensor networks will be able to leverage from mobility in sensors. For instance, using mobility; sensors can move towards coverage holes to enhance quality of initial deployment [4], [5], [6]; better routes for packets can be found [7]; data reliability can be enhanced if sensors move closer to events [8] etc. However, despite the advantages that mobility offers to sensor networks, there is one critical constraint on it that cannot be avoided. Sensors are severely energy constrained, and available energy has to be shared for sensing, data processing, transmission etc. Since mobility also consumes energy, it is very likely that there is a limit on the overall movement distance capability of the sensors. This is especially true in environments when sensors have to work un-attended after deployment, and where recharging sensors is not always feasible (e.g., hostile zones, battlefields etc.)

To validate our above claim on mobility limitations, we briefly discuss two recent mobile sensor models implemented in practice. Lymberopoulos and Savvides in [1] have designed a motion-enabled and power aware sensor node platform. A battery enabled miniature geared motor actuates sensor motion. The maximum movement distance of sensors in this design is $165\ meters$. In another development, as part of the self-healing minefield program, DARPA has designed a class of sensors with limited hop-by-hop mobility to detect and repair breaches in battlefields [2]. Mobility here is powered by fuel-propeller mechanism, and the sensors can make up to $100\ hops$. While the internal mobility semantics may be different in both models, the fact is that the sensor's maximum movement distance is limited.

In this paper, we address an important sensor networks deployment problem under mobility constraints on sensors. The sensor network in our problem 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 $\bar{k}$) that is application decided. In this scenario, our problem statement is; Given a deployment of limited mobility sensors in the network, the objective is to determine a sequence of sensor movements in order to minimize the variance in the number of sensors from $\bar{k}$ among all regions in the network, and simultaneously

minimize the total number of sensor movements.

Motivation: Our deployment problem is representative in many sensor network applications. The approach to cluster sensor networks into regions has been widely adopted in practice [9], [10], [11] and [12], as clustering enhances scalability, improves routing and power efficiency, provides better support to higher level functionality etc. The desired number of sensors ($\bar{k}$) per region can be contingent on one or more factors including sensing, fault tolerance, resilience to attacks, lifetime etc. For instance, due to physical limitations on sensors and associated hardware, the sampling rate at which sensors can sense the environment may be limited (e.g., $100\ kHz$ for acoustic sensors [13], and $4200\ Hz$ for magnetometers [14]). Thus in applications where the environment needs to be sensed quite frequently, or at all times of operation (e.g., intruder tracking, military surveillance etc.), multiple sensors per region have to be deployed to meet sensing objectives. Secondly, there may be obstacles in the regions or external factors (like heat, vibration etc.) that affect sensing ranges during network operation. Such sensing dynamics can be compensated with multiple sensors per region. Furthermore, fault tolerance, resilience to attacks improve with multiple sensors; lifetime can be prolonged using role rotation among multiple sensors etc. Hence, multiple sensors per region provide many benefits to sensor networks. However, the desired $\bar{k}$ sensors per region requirement may not be always satisfied. For example, if sensors are randomly deployed (sprayed from a vehicle, airdropped etc.), the requirement is hard to satisfy. Even if at initial deployment all regions have $\bar{k}$ sensors, as time goes on, faults, failures, energy losses etc. can violate this requirement. In such cases, the limited mobility sensors have to self-adjust their positions to correct such violations.

Recently, some works have appeared where sensor mobility is leveraged to enhance deployment [4], [5], [6] and [15]. However, the key shortcoming in such works is that sensor mobility limitations are not explicitly considered. Specifically, it is assumed that if a sensor wishes to move to a new location, it can do so without any restriction in its movement distance. However, as discussed in the sensor mobility instances above, this may not always be true. Under hard mobility constraints, existing works have limited applicability. For instance, in the well known virtual force approach [4], [6] and [15], sensors exert *virtual forces* among themselves. Two sensors repel (or attract) each other if they are too close (or too far apart). By balancing virtual forces, sensors spread themselves in the field. However, under hard mobility

constraints, two sensors may not be able to achieve force balance if the distance required to be traversed is too large. Secondly, the virtual force approach results in several back and forth movements during force balancing, which across many iterations will rapidly deplete sensor mobility capacity. For similar reasons, other works on mobility assisted deployment also have limited applicability under hard sensor mobility constraints (more discussions on the challenges of limited mobility appear in the next section). In this paper, we address the issue of limited mobility sensor networks deployment.

Contributions: In this paper, we design a set of movement algorithms for our deployment problem that can be executed by limited mobility sensors. Our contributions are:

A methodology for translating our non-linear optimization problem: Our first contribution is a weight-based methodology that translates our non-linear variance objective into a linear one. We propose a weight assignment rule for regions depending on $\bar{k}$, so that when sensors move, larger weights regions are given higher priority to balance sensor movements among all regions. We then define a new linear objective function called *Score* that captures weighted sensor movements, and prove that maximizing the *Score* minimizes the deployment *Variance* and vice versa. The number of sensor movements is minimized by treating each movement as a cost, and minimizing overall costs during *Score* maximization.

The Optimal maximum flow based centralized algorithm: Our first algorithm is the Optimal Maximum Flow based (*OMF*) centralized algorithm. Here, the sensor network at initial deployment is translated into a graph (G_V). Vertices in G_V represent regions, and are assigned appropriate weights based on the above methodology. Edges represent movement ability between regions, and are assigned corresponding capacities and costs. We first show how the *minimum cost maximum weighted flow plan* in G_V maximizes the *Score* with minimum cost. We then show how this flow plan can be translated as a sensor movement plan that minimizes deployment *Variance* and sensor movements in the network. 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 is to maximize the summation of the flow amount to each target multiplied with the target weight. The maximum flow problem is its special case, where the weight of each target is one.

The Simple Peak-Pit based distributed algorithm: We then propose a local, light-weight and purely distributed Simple Peak-Pit based (*SPP*) algorithm that is executed by sensors themselves. Regions needing

sensors send local requests containing *weights* based on number of sensors needed. Surplus regions that receive the requests will serve them in a descending order of weights, along with minimizing sensor movements in serving them. As discussed subsequently, path feasibility (to guarantee unbroken chain of movements) is ensured in our algorithms before sensors make real movements.

Theoretical analysis and Performance evaluations: We conduct a detailed theoretical analysis and performance evaluations of our algorithms. We formally prove the optimality of our *OMF* algorithm in minimizing variance and number of sensor movements, and derive its complexity. We then conduct extensive simulations to evaluate the performance of our algorithms. For comprehensiveness, we also simulate the well known Virtual Force algorithm [4]. In general the *OMF* algorithm (being optimal), achieves best variance and sensor movement minimization. However, under certain scenarios (small $\bar{k}$, uniform initial deployment), performance of the *SPP* algorithm is close to the *OMF* algorithm. We also study communication overhead in our algorithms. We observe that the overhead in the *SPP* algorithm is generally lower. However, when initial deployment is highly concentrated, the overhead in the *OMF* algorithm is quite close to the *SPP* algorithm, while being smaller in some cases. Finally, we observe that all our algorithms have better performance than the virtual force algorithm, with less overhead. As pointed before, this is due to many back and forth sensor movements in the virtual force algorithm resulting in rapid expiration of sensor mobility capacity.

Our paper is organized as follows. In Section II, we formally define our deployment problem, and detail the methodology of our proposed algorithms in Section III. In Section IV, we present our *OMF* algorithm and prove its optimality. In Section V, we present our *SPP* algorithm and its features. We present some discussions in Section VI, and performance evaluations in Section VII. Related work is presented in Section VIII, and we conclude our paper in Section IX.

II. Our Sensor Network Deployment Problem

A. Problem Definition

Our sensor network 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 a sensor is capable of. 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, we want to determine a sequence of sensor movements so 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 also 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.

Problem Features: Our problem is general, since we place no restriction on $\bar{k}$. If $\bar{k} = 1$, then the requirement is one sensor per region. To enhance reliability, $\bar{k}$ can be set larger than 1. Also, it is not necessary that $\bar{k}$ is same for all regions. In non-uniform environments, some regions may need more sensors than others, meaning $\bar{k}$ is different for different regions. The *Variance* definition still holds, except that $\bar{k}$ in equation (1) becomes $\bar{k}_i$ for region i. Our problem is also not contingent on the number of mobile sensors. It holds even when only a part of sensors in the network are mobile [1].

An important feature of our problem is that we do not minimize the variance in 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

[1]In the following, we discuss solutions for the basic problem first. Extensions to other problems are discussed later.

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.

Assumptions: We make the following assumptions. We assume that $min\{\frac{S_{sen}}{\sqrt{2}}, \frac{S_{tr}}{\sqrt{5}}\} \geq R$, where R is the region size, and S_{sen} and S_{tr} are a sensor's sensing and transmission ranges respectively. If each region has $\bar{k}$ sensors at final deployment, then $\frac{S_{sen}}{\sqrt{2}} \geq R$ means every point in each region is *covered* by $\bar{k}$ sensors, and $\frac{S_{tr}}{\sqrt{5}} \geq R$ means a sensor in any region can communicate with $\bar{k}$ sensors in each of its four adjacent regions. We also assume sensors are homogeneous in sensing and transmission ranges, and they are unaffected during network operation as in [4], [5], [6], [15]. We assume a free space radio propagation model, where there exists a clear line of sight path between two communicating sensors in the network. We assume that each sensor knows which region it resides in. To do so, sensors can be provisioned with GPS devices, or methods in [16] can be used, where sensor location are determined 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 this case, the general case where a sensor can move in any arbitrary direction is discussed next. Also, we first 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 network, 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 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 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 that region.

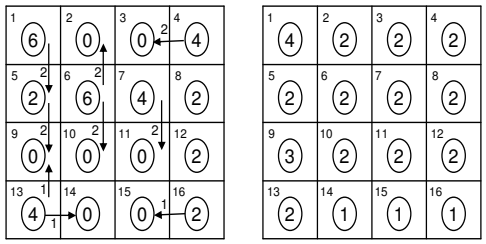

Fig. 1. An instance of initial deployment and an intuitive movement plan to minimize variance (a), and the resulting deployment (b).

Let us denote regions that have at least *one* sensor at initial deployment as *source regions* (or *sources*), and 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 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 final 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) cannot 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 $H = 1$. 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 movement plan that can make the variance 0. If so, what is the plan, or more importantly, what are the challenges need to be addressed in this movement plan. We discuss both issues below.

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

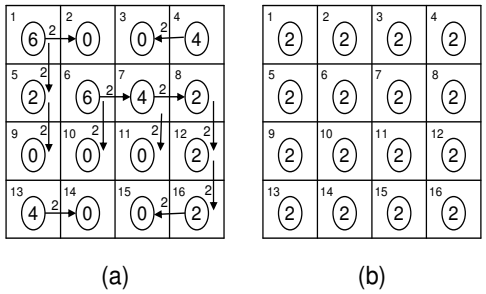

Fig. 2. An instance of initial deployment and an optimal movement plan (a), and the resulting deployment (b).

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 are,

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

Note that sink s_i^m has more weight than s_i^n, if $m > n$. Also, $w_i^m = w_j^m$ 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. (Please see Appendix for proof.)

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

Our first algorithm is the Optimal Maximum Flow based (*OMF*) centralized 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.

A. Description of the Algorithm

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 VI-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.

1) Steps in Algorithm Execution: 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 [17], [18], [9], [19] 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 [16] 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), 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.

2) 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.

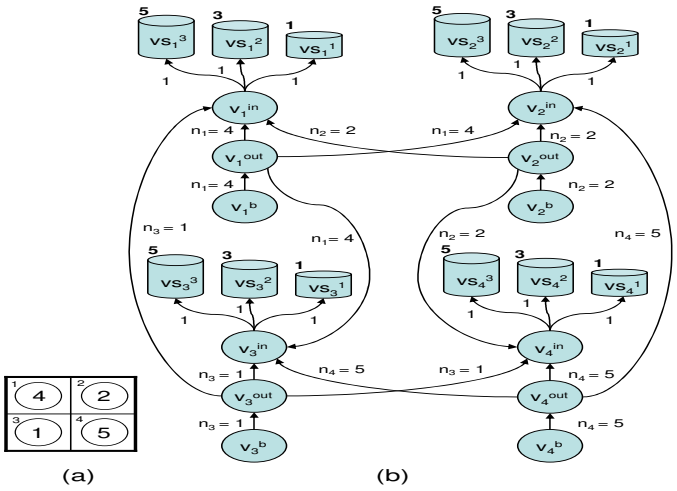

Fig. 3. An instance of the initial network deployment (a) and the corresponding virtual graph G_V (b).

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-A.4). 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.

3) 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

1: Input: $G_V(V_V, E_V)$, H, S and $\bar{k}$
2: Output: Graph $G_V^m(\bar{V}, \bar{E})$ and Minimum Cost Maximum Weighted Flow Plan Z in G_V
3: $|E_V|$ = No. of Edges in G_V, $|V_V|$ = No. of Vertices in G_V
4: $|\bar{E}| = |E_V| + S \times (\bar{k} + 1)$
5: $|\bar{V}| = |V_V| + 2$
6: Add vertices S^v_{source} and S^v_{sink} to G_V to create graph G_V^m
7: **for** each region i **do**
8: **for** j from 1 to $\bar{k}$ **do**
9: Add edge from sink vs_i^j to S^v_{sink}
10: Assign corresponding edge capacity as 1
11: Assign corresponding edge cost as $-(2j-1) \times H \times |\bar{E}|$
12: **end for**
13: Add edge from S^v_{source} to v_i^b
14: Assign corresponding edge capacity as ∞
15: Assign corresponding edge cost as 0
16: **end for**
17: Determine the maximum flow value $|\bar{Z}|$ from S^v_{source} to S^v_{sink} in G_V^m
18: Determine the minimum cost flow plan Z (for flow value $|\bar{Z}|$) from S^v_{source} to S^v_{sink} in G_V^m

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_i^1, vs_i^2, vs_i^3, ..., $vs_i^{\bar{k}}$ 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 [2]. Finally, edges are added from S^v_{source} to all base vertices (i.e., v_i^b 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_V^m has been constructed. Determining the flow plan to maximize weighted flow to sinks with minimum cost in G_V^m 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_V^m. The maximum flow value $|\bar{Z}|$ indicates the maximum number of sinks that can get a sensor in G_V^m. 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

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

priority. Our objective however, is to determine the *flow plan* Z (the actual flow among the edges) in G_V^m 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_V^m, 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_{sink}^v 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_{sink}^v. There is one issue we have to resolve during cost assignment. Recall that sensor movements between reachable regions are considered as costs in G_V^m. Clearly, these costs will affect the minimum cost flow plan when determining flows to sinks with minimum cost in G_V^m. To prevent this from happening, the costs from sinks to S_{sink}^v 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_V^m 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_V^m and G_V.

Theorem 2: The flow plan corresponding to the minimum cost maximum flow in G_V^m is the flow plan corresponding to the minimum cost maximum weighted flow in G_V (Please see Appendix for proof).

4) 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 where to, is forwarded by the Base-station to the sensors in the network.

B. Optimality of our OMF *Algorithm*

Before discussing optimality, 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 sensor movement sequence.

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 (Please see Appendix for Proof).

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 (For proof please refer to [20]).

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_{opt}^V be the minimum cost maximum weighted flow plan in G_V. Its corresponding movement plan Z_{opt}^S will minimize variance and the number of sensor movement hops in the sensor network (Please see Appendix for Proof).

We now discuss time complexity of the *OMF* algorithm. There are three phases in our algorithm in determining the optimal movement plan. The first is construction of $G_V(V_V, E_V)$ and $G_V^m(\bar{V}, \bar{E})$, the second is determining the maximum flow in G_V^m, and the third is determining the minimum cost flow in G_V^m. The time complexity is dominated by determining the maximum flow and minimum cost flow in G_V^m. Our implementations of the maximum flow algorithm is the Edmonds-Karp algorithm [21], and minimum cost flow algorithm is the one in [22]. The resulting time complexity 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_V^m, 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.

V. The Simple Peak-Pit based Distributed algorithm

In the above, we presented a centralized and optimal *OMF* algorithm to our deployment using our weight-based methodology. We now present the Simple Peak-Pit (*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 event driven. Using inter-region communications, a region leader will be elected for co-ordination. Each 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 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 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

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
```

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 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 [3]. Sensors will start moving after $ACKs$ are sent, following the reserved paths of the corresponding $REQs$.

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

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 terminates when each pit has either obtained at-least $\bar{k}$ sensors, or expiration of movement capability of the sensors, or if a certain number of requests have been tried without success by requesting sensors [4].

It may happen some regions have no sensors in them (i.e., *holes*) after initial deployment. A hole can be filled by one of its neighbors with extra mobile sensors, after coordination by 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 get sensors during the *SPP* algorithm execution

VI. Discussions

In the above, we presented an optimal centralized *OMF*, and a distributed *SPP* algorithm for our deployment problem. We now discuss execution of our algorithms, extensions to non-uniform scenarios, the issues of arbitrary sensor movement directions and network partitions.

A. Executing our Algorithms

The algorithms we proposed above can be executed in more than one way. We first discuss a semi-distributed version of the the *OMF* algorithm, called the *Domain-based OMF* (*D-OMF*) algorithm. Here 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. Note that the *D-OMF* algorithm being semi-distributed has lower messaging and computational complexity than *OMF* algorithm. But, optimality is compromised since the *D-OMF* algorithm achieves local optima in each domain and cannot guarantee

[4]The number of unsuccessful requests per sensor is application decided.

global optima. Note that this trade-off depends on the domain size, uniformity of initial deployment and sensor mobility capacity. Conducting an analytical comparison of performance of *D-OMF* and *OMF* algorithms is too difficult if not impossible. We study this using extensive simulations in Section VII. Furthermore, we point out that our proposed algorithms can also be combinedly executed. A simple instance is one where the *OMF* algorithm is executed first to optimize deployment, and at later stages the distributed algorithms can be executed to repair deployment under faults, failures etc.

B. Extensions to Non-Uniform Scenarios

1) Non-Uniformity at Sensor Side: We have so far assumed that all sensors are homogeneous in their mobility capacity. In many scenarios, due to deployment costs, faults in sensors etc., it may happen that only a subset of the deployed sensors is mobile. Our solutions can be extended in such scenarios. The weight assignment rule is still the same. In the *OMF* algorithm, we have to modify reachability information in G_V. For example, say only one sensor in region 2 in Figure 3 is mobile. Then the edges from region 2 to its reachable regions 1 and 4 (i.e., from v_2^{out} to v_1^{in}, and from v_2^{out} to v_4^{in}) each have capacity one. This allows upto only one sensor to move out from region 2. Other construction rules remain the same. The resulting solution is still optimal. The *SPP* algorithm needs no changes. Only, fewer paths will be feasible now due to not all sensors being mobile.

2) Non-Uniformity at Deployment Area Side: In our discussions above, we focused on uniform deployment areas, where $\bar{k}$ is the same for all regions. However, in many situations the deployment area can be non-uniform. Examples are certain sensitive zones that need to be sensed to a higher degree, which means $\bar{k}$ is more in such zones that others; certain hostile zones like lakes, fires etc. that can destroy sensors, which means $\bar{k} = 0$ for such zones etc. For addressing such requirements, our weight assignment rule is still the same as in equation (3). However, the number of sinks created per region and their corresponding weights will be different depending on the desired $\bar{k}$ per region in the *OMF* algorithm. In the *SPP* algorithm, the number of requests generated and their weights are modified accordingly. The rest of our solutions is still the same.

C. 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 the case of arbitrary sensor movement directions. For *OMF* and *D-OMF* algorithms, only virtual graph (G_V) construction 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 movement directions. In *SPP* algorithm, there are now more neighbor choices to forward a request, and extra feasible paths can be reserved while sensors move to satisfy requests.

In Section II, we assumed that the sensor network is not partitioned. In some situations it may happen that 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 [5], empty holes are filled by placing a *seed* from a non-empty region to a hole. We can apply the algorithms in [5] 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.

VII. Performance Evaluations

In this section, we report our experimental data to study the performance of our *OMF*, *D-OMF* and *SPP* algorithms under various sensor and network parameters. We also simulate the well known VORonoi-based Virtual Force (*VOR*) algorithm proposed in [4] and compare its performance with our algorithms.

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 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 $n \times n \times \bar{k}$, and all sensors in the network are mobile by default. For the *D-OMF* algorithm, we choose the domain size D as $D = \frac{n}{2}$. Our implementations of the maximum flow algorithm is the Edmonds-Karp algorithm [21], and minimum cost flow algorithm is the one in [22]. 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. For comparisons, we also simulate the VORonoi-based virtual force (*VOR*) algorithm [4], the basic idea of which was discussed in Section I. The termination condition for the *SPP* algorithm is when each pit has either obtained at-least $\bar{k}$ sensors, or expiration of sensor movement capability, or if a certain number of requests have been tried without success by requesting sensors. By default, the number of requests per sensor without success is set as 3. For the *VOR* algorithm, the termination condition was

each region obtaining at-least $\bar{k}$ sensors or expiration of sensor movement capability.

We conduct our simulations on a custom simulator [5]. For initial deployment, our simulator uses a topology generator for 2D-Normal distribution [23]. A 2D-Normal distribution involves two random variables, x and y with mean values μ_x and μ_y. The mean values corresponding to each variable can be written as a vector $\mathbf{u} = (\mu_x, \mu_y)^T$. Each variable will have a variance σ_x and σ_y. However, it may happen that the variables are related to each other, in which case there will be covariances σ_{xy} and σ_{yx} with $\sigma_{xy} = \sigma_{yx}$, all of which can be incorporated into a variance-covariance matrix: $\mathbf{v} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{yx}^2 & \sigma_y^2 \end{pmatrix}$. The 2D-Normal distribution is then given by $P(\mathbf{z}) = \frac{1}{2\pi\sqrt{|\mathbf{v}|}} e^{[-\frac{1}{2}(\mathbf{z}-\mathbf{u})^T \mathbf{v}^{-1}(\mathbf{z}-\mathbf{u})]}$ where $|\mathbf{v}|$ is the determinant of $\mathbf{v}$. In our simulations, we set $\sigma_{xy} = \sigma_{yx} = 0$, which means the location at x and y axis are independent when sensors are deployed. We let $\sigma^2 = 1/\sigma_x^2 = 1/\sigma_y^2$. Hence, when σ increases, sensors will be more concentrated at the center, and when σ tends to 0 sensors are more uniformly distributed. For our simulations, by default, $\sigma = 4$. All data reported here were collected across 10 iterations, and averaged [6].

B. Performance Results

1) Performance comparison of all algorithms: We first study the sensitivity of VI, MH and PN to mobility capacity H for the *OMF*, *D-OMF*, *SPP* and *VOR* algorithms. We assume that the sensors are initially deployed as a one time step targeted towards the center of the network. All other settings are default. From Figure 4, 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 performance of the *SPP* and *VOR* algorithms are quite good for smaller values of H. Among all algorithms, the virtual force (*VOR*) algorithm has the poorest performance. Since sensors in the *VOR* algorithm attempt to achieve local force balance between themselves, they incur several back and forth movements that rapidly depletes overall sensor movement capability resulting in overall poor variance improvement. Figure 4 also shows the effects of limited mobility on VI. When H increases, VI increases in all algorithms as increased movement ability in general helps to move sensors to needy regions farther away. The improvement stays constant when $H \geq 4$ for the *OMF* algorithm, and for $H \geq 3$ in the other

[5]Code can be obtained by contacting the primary author of the paper.

[6]The standard deviations for all data sets are shown in the corresponding figures.

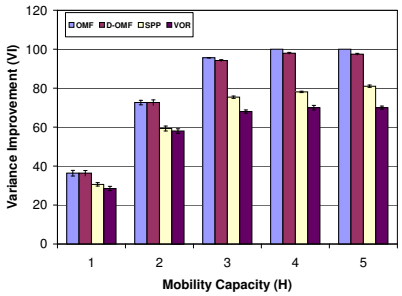

Fig. 4. Sensitivity of VI to H for all four algorithms

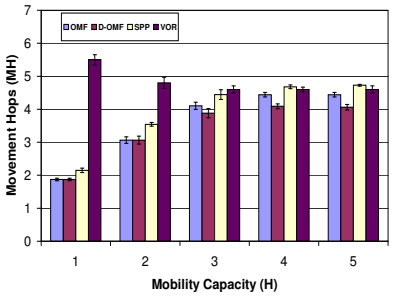

Fig. 5. Sensitivity of MH to H for all four algorithms

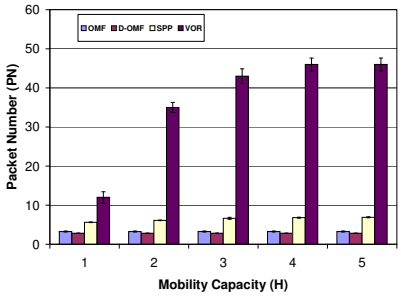

Fig. 6. Sensitivity of PN to H for all four algorithms

algorithms. This demonstrates that mobility beyond a certain point does not bring in further benefits. Note that when $H \geq 4$, the *OMF* algorithm achieves the upper bound of 100% VI, because movement choices can be *optimally* exploited by the *OMF* algorithm with larger H, unlike other algorithms.

Figures 4 and 5 show that when VI increases, MH also increases for the *OMF*, *D-OMF* and *SPP* algorithms. In our algorithms, each sensor movement typically results in a more balanced deployment for the network. Also, it could happen that for improvement in variance, many potentially longer paths are found in our algorithms. Consequently, when VI increases MH also increases in our algorithms, while staying a constant when VI stops increasing. On the other hand, we observe that the MH for the *VOR* algorithm is initially a higher value, then decreases and stays constant. When H is small, VI is quite low for the *VOR* algorithm, due to more stringent mobility limitations, which makes MH higher for smaller H. As VI improves further, MH decreases for the *VOR* algorithm in Figure 5. Since VI stays constant beyond $H > 2$, MH also stays constant. Note that when H increases, the MH in the *D-OMF*, *SPP* and VOR algorithms are quite close to the optimal *OMF* algorithm (while even being smaller in some cases). As pointed before, this is because of the improved VI that can be achieved by the *OMF* algorithm, by exploiting several additional movement choices compared to the other algorithms, which increases the number of movements, and hence MH in the *OMF* algorithm.

In Figure 6, 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. The messaging overhead in the VOR algorithm is the maximum because of many local message exchanges caused by several back and forth sensor movements. The PN in *SPP* algorithm takes a middle ground, since only deficient regions send out requests and that too towards richer regions only, while responses always aim to take shorter movement paths. In both *SPP*

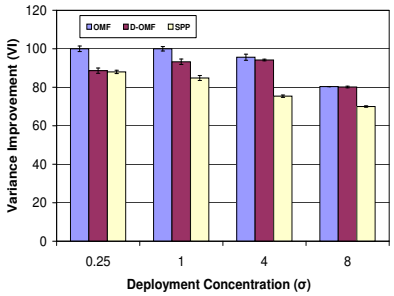

Fig. 7. Sensitivity of VI to σ

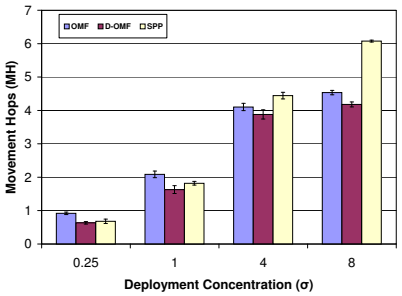

Fig. 8. Sensitivity of MH to σ

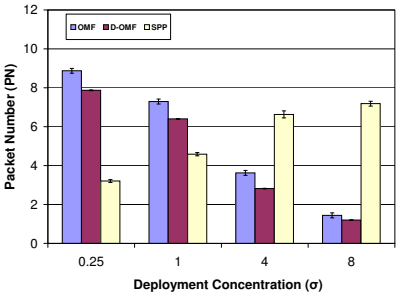

Fig. 9. Sensitivity of PN to σ

and VOR algorithms, PN increases with H due to more movement choices when H increases.

Figures 7, 8 and 9 show the sensitivity of our performance metrics to σ for our algorithms. In Figure 7, 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.

In Figure 8, 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 9, 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 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

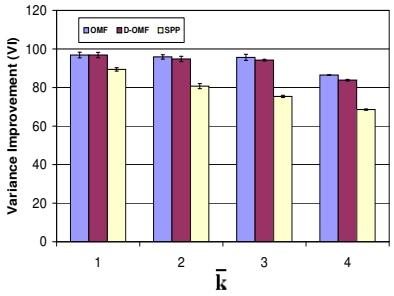

Fig. 10. Sensitivity of VI to $\bar{k}$

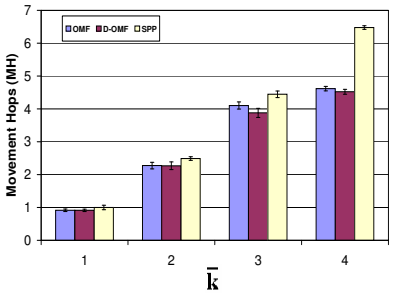

Fig. 11. Sensitivity of MH to $\bar{k}$

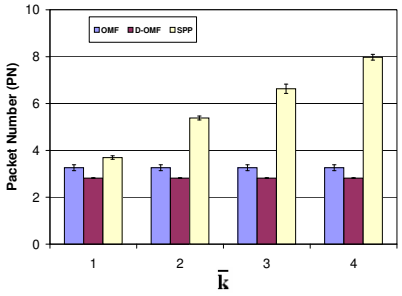

Fig. 12. Sensitivity of PN to $\bar{k}$

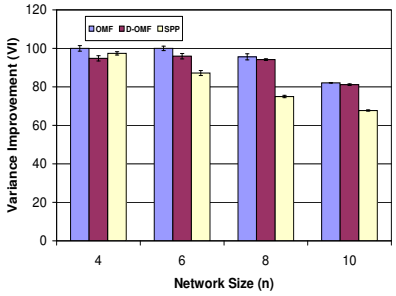

Fig. 13. Sensitivity of VI to n

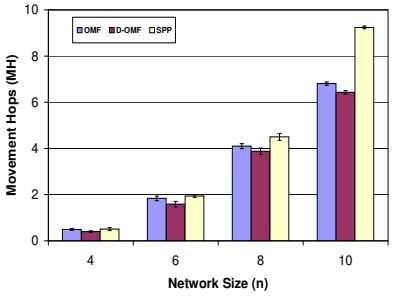

Fig. 14. Sensitivity of MH to n

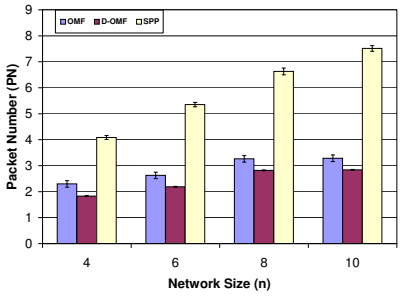

Fig. 15. Sensitivity of PN to n

cases (all other settings are default). From Figures 10 and 11, 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 12 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 13, 14 and 15, 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.

2) Performance when only a subset of sensors are mobile: Our default case above consisted of all sensors in the network as capable of being limitedly mobile. We now demonstrate the sensitivity of

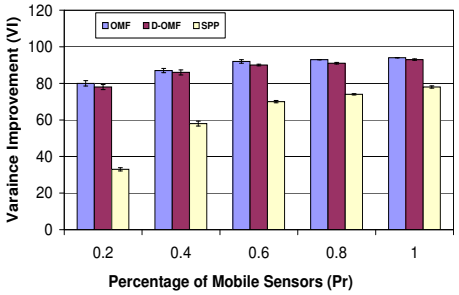

Fig. 16. Sensitivity of VI to Pr

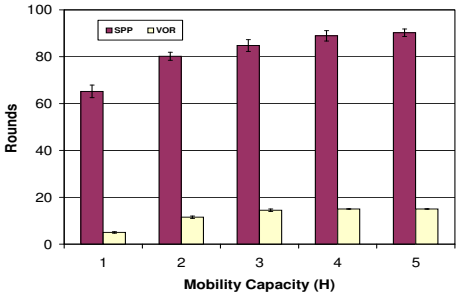

Fig. 17. Sensitivity of convergence time to H in the *SPP* and *VOR* algorithm

performance (VI) of our algorithms under different sensor mobility capacity (H) when only a subset of sensors is mobile (as discussed in Section VI-B.1). The value of H is set as 3. All other settings are default. In Figures 16, term P_r on the X-axis denotes the percentage of sensors that are mobile. For instance if $P_r = 0.2$, then only 20% of the sensors are mobile. We observe that while VI improves with increasing P_r, there is a threshold beyond which increase in VI is negligible in all agorithms. For the case when $H = 3$ in Figure 16, the threshold is around 60%. The threshold in fact depends on H and decreases as H increases and vice versa [7]. This demonstrates that, not all sensors in the network need to be mobile. Depending on the mobility capacity H, there is a threshold beyond which deployment quality cannot be enhnaced significantly with more mobile sensors.

3) Convergence time of the SPP *and* VOR *algorithms:* In Figure 17, we study the sensitivity of convergence time of the *SPP* and *VOR* algorithms to H. All other settings are default. For the *SPP* algorithm, the convergence time (in terms of rounds) is obtained as follows. We denote one unit time as the time taken by a sensor to perform local computations and send a packet to a sensor in a neighboring region. The number of rounds is the total number of unit times spent to complete execution of the *SPP* algorithm in the network. For the *VOR* algorithm, it is simply the number of rounds it takes for the algorithm to terminate similar to the definition in [4]. From Figure 17, we see the convergence time increases with H for both algorithms, since more movement choices are available with increasing H. We also observe that the convergence time begins to saturate with increasing H, demonstrating that beyond a certain point, increase in mobility does not help deployment much. Note that the number of rounds in the *VOR* algorithm is much lower than the *SPP* algorithm. However, this should not be construed as better

[7]We do not report data for other values of H due to space limitations.

performance by the *VOR* algorithm. Rather, it is due to the faster depletion of sensor mobility capacity, and the lower VI in the *VOR* algorithm compared to the *SPP* algorithm. The sensitivity of number of rounds to $\bar{k}$, σ and n follow expected trends and are not reported here (due to space limitations).

VIII. Related Work

In this paper, we addressed a sensor networks deployment problem using limited mobility sensors. While a host of works have appeared on deployment [4], [6], [15], [5], [24], [8], [25], [26], [27], [28], [29], [30], [31], [32], [33], in this section, we particularly discuss related work in the areas of mobility assisted sensor networks deployment.

In [4], [6] and [15], the goal is uniform coverage of the network. This means that every point in the network is covered by at least *one* sensor. The approach in [4], [6] and [15] is balancing sensor virtual forces. Two sensors may repel or attract each other based on the distance between them. At each iteration, sensors move to achieve a better force balance, and sensors stop moving when a force equilibrium is reached. However, under hard mobility constraints, two sensors may not be able to achieve force balance if the distance required to be traversed is too large. Secondly, the virtual force approach will result in several back and forth sensor movements during force balancing, which across many iterations will rapidly deplete mobility capacity of 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 [5], where the objective is load balancing sensor deployment. In [5], 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 [5] 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 [5] is a special case of our general deployment problem in this paper.

Other works in this area include [24], where algorithms are proposed to let sensors relocate to new positions in a 2-D dimensional grid based network. The relocation process is event-driven (new events, sensor failures, faults etc.). However, such relocation is done without compromising existing functionality

of the network. In [8], algorithms are designed to enable a sensor to move to events in the field. The algorithms are designed to be less energy consuming and computationally mild for the sensors. Both the above works are event-driven and also do not consider hard mobility limitations on sensors.

We have done some prior work in limited mobility deployment in [25]. There, our problem was maximizing number of regions in the network with at least one sensor, where the sensors were can *hop* only *once* to a *fixed* distance. The problem we address in this paper is minimizing variance, which is a non-linear objective. The corresponding methodology and algorithms in this paper are different from [25]. Also, we have a more general limited mobility model in this paper, where only the maximum sensor movement distance is limited. This paper also proposes a distributed algorithm unlike our work in [25].

IX. Final Remarks

In this paper, we defined a general sensor networks deployment problem under limited mobility sensors, and proposed a set of sensor movement algorithms for it. Our on-going work addresses the issue of repairing network partitions with limited mobility sensors. Also, we plan to study the issues of limited mobility sensors in applications like sensor tracking systems. The challenge is how to design algorithms that can exploit limited mobility in sensors, and algorithms for provisioning sensors in the network to improve tracking efficiency throughout the network and some specific hot spots. Finally, we are planning to investigate opportunities and challenges associated with mobility in more complex environments like those where link qualities, sensing ranges, transmission ranges are non-uniform.

References

[1] 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.

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

[3] K. Dantu, M Rahimi, H. Shah, S. Babel, A. Dhariwal, and G. Sukhatme, "Robomote: Enabling mobility in sensor networks," in *In Proceedings In IEEE/ACM International Conference on Information Processing in Sensor Networks (IPSN-SPOTS)*, Los Angeles, April 2005.

[4] 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.

[5] 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.

[6] 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.

[7] W. Wang, V. Srinivasan, and K. Chua, "Using mobile relays to prolong the lifetime of wireless sensor networks," in *Proceedings of ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Cologne, September 2005.

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

[9] 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.

[10] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of Hawaii International Conference on System Sciences (HICSS)*, Maui, January 2000.

[11] T. Abdelzaher, B. Blum, Q. Cao, D. Evans, J. George, S. George, T. He, L. Luo, S. Son, R. Stoleru, J. Stankovic, and A. Wood, "Envirotrack: An environmental programming model for tracking applications in distributed sensor networks," in *In Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, March 2004.

[12] C. Devaraj, M. Nagda, I. Gupta, and G. Agha, "An underlay for sensor networks: Localized protocols for maintenance and usage," in *Proceedings of IEEE International Conference on Mobile AdHoc and Sensor Systems (MASS)*, Washington D.C., November 2005.

[13] D. Eickstedt and H. Schmidt, "A low-frequency sonar for sensor-adaptive, multistatic, detection and classification of underwater targets with auvs," in *Proceedings of Oceans Vol. 3, Pages 1440-1447*, 2003.

[14] H-P Mller, R Rossi A Pasquarell and, M De Melis, L Marzetti, A Trebeschi, and SN Ern, "Argos 500: Operation of a helmet vector-meg," in *Neurology, Neurophysiology and Neuroscience*, 2004.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," in *Proceedings of International Workshop on Discrete Algorithms and methods for mobile computing and communications*, Seattle, August 1999.

[20] S. Chellappan, W.Gu, X. Bai, B. Ma, D. Xuan, and K.Zhang, "Deploying wireless sensor networks under limited mobility constraints," Tech. Rep. OSU-CISRC-9/05-TR58, Dept. of Computer Science and Engineering, The Ohio-State University, September 2005.

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

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

[23] J. Patel and B. Campbell, *Handbook of the Normal Distribution - 2nd edition*, CRC, 1996.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

APPENDIX

Proof of Theorem 1 in Section III:

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.

■

Proof of Theorem 2 in Section IV-A.3:

Proof: We first prove that the flow plan corresponding to the minimum cost maximum flow in G_V^m 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. ■

Proof of Lemma 1 in Section IV-B:

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-A.2). 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. ■

Proof of Theorem 3 in Section IV-B:

Proof: We first prove that our *OMF* algorithm is optimal in terms of minimizing variance. We prove by contradiction. Consider a sensor movement plan Z_{opt}^S that corresponds to a flow plan Z_{opt}^V 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 there is a better movement plan, Z_x^S that can further minimize variance in the sensor network. By Corollary 1, a corresponding flow plan Z_x^V can be found in G_V. The amount of weighted flow in this plan is larger than the weighted flow achieved using plan Z_{opt}^V, which is a contradiction. Hence Z_{opt}^S is the optimal movement plan for sensors that minimizes variance.

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