

# Connectivity-Guaranteed and Obstacle-Adaptive Deployment Schemes for Mobile Sensor Networks\*

Guang Tan<sup>†</sup>  
INRIA-Rennes  
35043 Rennes, France  
gtan@irisa.fr

Stephen A. Jarvis  
Dept. of Computer Science  
The University of Warwick, UK  
saj@dcs.warwick.ac.uk

Anne-Marie Kermarrec  
INRIA-Rennes  
35043 Rennes, France  
akermarrec@irisa.fr

## Abstract

*Mobile sensors can move and self-deploy into a network. While focusing on the problems of coverage, existing deployment schemes mostly over-simplify the conditions for network connectivity: they either assume that the communication range is large enough for sensors in geometric neighborhoods to obtain each other's location by local communications, or assume a dense network that remains connected. At the same time, an obstacle-free field or full knowledge of the field layout is often assumed. We present new schemes that are not restricted by these assumptions, and thus adapt to a much wider range of application scenarios. While maximizing sensing coverage, our schemes can achieve connectivity for a network with arbitrary sensor communication/sensing ranges or node densities, at the cost of a small moving distance; the schemes do not need any knowledge of the field layout, which can be irregular and have obstacles/holes of arbitrary shape. Simulations results show that the proposed schemes achieve the targeted properties.*

## 1. Introduction

In a mobile sensor network, the sensors can move and self-organize into a network. The mobility and self-management of sensors are desirable for many application scenarios such as remote harsh fields, disaster areas, or toxic urban regions, where manual operations are unsafe or burdensome. In this paper we consider the following self-deployment problem: *given a target sensing field with an arbitrary initial sensor distribution, how to guide sensors to self-organize into a connected network that has the maximum coverage at the cost of the minimum moving distance?*

There have been a number of proposals for this problem. One of the main approaches is based on the concept

of *potential fields* [5] or *virtual forces* [13]. This approach imitates the behavior of electro-magnetic particles: when two electro-magnetic particles are too close to each other, a repulsive force pushes them apart. Applied to a sensor network, this method helps to move sensors from high density areas to low density areas, thereby minimizing sensing overlap and improving the overall network coverage. Another commonly used approach relies on the use of Voronoi Diagrams (VDs) [2, 4] to partition the field into many sub-areas, one for each sensor, so that the sensors can move to maximize coverage in its own sub-area. A combination of these two approaches is also possible; see [9].

While the previously proposed methods prove to be effective under certain models, they have several problems in practice. First, they assume that a sensor can easily detect all (or most) of its Voronoi neighbors through local communication. This assumption however is not necessarily satisfied in a real network, because the communication range of a sensor may not be large enough to cover all those neighbors. The incomplete view of the Voronoi neighbors may result in very inaccurate VDs being constructed. Consequently, significant sensing overlaps or voids among sensors may be ignored, leading to poor network coverage. Second, previous studies, while concentrating on the motion planning of sensors, tend to assume that the network remains connected throughout the process of sensor relocation. The implicit assumption that the network connectivity can be guaranteed by a high node density, or a large communication range, however, does not hold in general, for network partition can still happen in a very dense network. Finally, when planning movement for increasing coverage, most previous studies assume that the sensing field is obstacle-free (e.g., [8, 12]). This greatly simplifies the motion planning and optimization. However, many real-world environments, such as a metropolitan area with buildings, a rural environment with plants and water, naturally have obstacles or holes; this makes these schemes no longer applicable to such applications.

In this paper, we present new schemes that do not

\*This research was sponsored in part by the Engineering and Physical Sciences Research Council (EPSRC) UK, project number EP/F000936/1.

<sup>†</sup>The author's research was largely done while he was at the University of Warwick.

need those simplifying assumptions, and thus adapt to a much wider range of application scenarios. While maximizing sensing coverage, our distributed schemes aim to achieve three additional goals: (1) to achieve connectivity for a network with arbitrary initial distribution, communication/sensing ranges, or node densities. (2) to minimize moving distance, which dominates energy consumption in the deployment process. (3) to be able to work without any knowledge of the field layout, which can be irregular and have obstacles of arbitrary shape. This is important for deployment in an environment whose layout is not fully known in advance.

Our first solution is called the *Connectivity-Preserved Virtual Force (CPVF)* scheme (Section 4), which is an enhanced form of the virtual-force-based method with additional consideration of the connectivity requirements under generalized network settings. We show that the localized communication, which is the very reason for its simplicity, results in poor coverage in certain cases. We then consider a *Floor-based* scheme (or FLOOR for short) which overcomes all the difficulties of CPVF and significantly outperforms it (Section 5). Finally, we validate our approaches through simulation and examine the performance of the two schemes in Section 6.

## 2. Related Work

The authors in [11] show that when the communication range  $r_c$  is at least twice the sensing range  $r_s$ , then coverage implies network connectivity. Bai et al. [1] first present a full picture of optimal deployment patterns that achieve both coverage and connectivity for general values of  $r_c/r_s$ . Their analysis only considers an obstacle-free plane, therefore the results do not apply to a field with holes of general shape.

In a mobile sensor network, the mobility of sensor nodes provides the possibility for the nodes to self-organize to a network with desired properties from an arbitrary initial distribution. Howard et al. [5] employ the potential field method in sensor deployment, assuming a *line-of-sight* connectivity, that is, a sensor is always able to determine the locations of nearby nodes. This actually requires special sensing ability or alternatively, a large communication range, in a non-dense network. Zou. et al. [13] propose a VF-based deployment scheme, which is centralized and only works for a small cluster.

In [9], Wang et al. present a set of VD-based schemes to maximize coverage. The VDs are used by sensors to detect “coverage holes” in their vicinities, and sensor locations are adjusted round by round so that the coverage is gradually improved. Other schemes that have employed VDs include [2, 4, 9, 10]. As mentioned before, this approach implicitly relies on a relatively large  $r_c/r_s$  to guarantee the construction of a correct VD. Furthermore, their studies do not con-

sider the connectivity and obstacle problems.

Yang et al. [12] use a grid-based network structure to detect low density areas, and then maximize sensing coverage through balancing the sensor distribution. The same structure is also used in [8] to identify redundant sensors. Unfortunately, a grid structure is feasible only for networks with relatively high densities, which, again, translate to a relatively large  $r_c$ . In addition, both sets of research require an obstacle-free field.

## 3. Preliminaries

**System assumptions** We assume that all sensors have the same communication range  $r_c$  and sensing range  $r_s$ ; sensors within  $r_c$  of a sensor are called that sensor’s neighbors. As is the case for many applications, a sensor can determine another sensor’s location only by communication. (Assuming also location determination by sensing only makes the design simpler). At any given time, a sensor knows its own position and can recognize the boundary of the obstacles within its sensing range. Sensors move in *steps* of variable distance. In each step, a sensor moves in a straight line at a uniform speed for a fixed amount of time (e.g., 2 seconds), which we call a *period* and denote by  $T$ ; and at the end of that step, it decides the direction and size of the next step, and so on. The maximum moving speed is denoted by  $V$ .

We assume that the field is on a 2-D coordinate plane, which can contain any number of obstacles of arbitrary shape, as long as the field is connected; that is, any two points in the non-obstacle areas of the field can be connected by a continuous path. There is a reference point  $O$  known to all sensors, where  $O$  could be the location of the base station or some head sensor; all the sensors will try to connect to  $O$  either directly or via multi-hop links. Without loss of generality, we assume  $O$  is at  $(0, 0)$ .

**Obstacle avoidance algorithm** We use Lumelsky and Stepanov’s path planning algorithm [7] called “BUG2” to help a sensor move from a starting point *Start* to a destination point *Target*. Roughly speaking, this algorithm works as follows. The sensor initially moves along the straight line  $(Start, Target)$ , which we call the *reference line*, until it encounters an obstacle at some *hitting point H*; then, the sensor follows the boundary of the obstacle using the right-hand rule (i.e., the right hand maintains contact with the obstacle), until it gets back to the reference line at some point. Now, if the sensor finds that it is closer to the *Target* than from  $H$ , and that it can make progress on the reference line, then it resumes its straight line walk toward the *Target*; otherwise it continues to navigate around the obstacle. The above procedure repeats until the sensor reaches *Target*. For more details of the algorithm the reader is referred to [7].

**Lazy movement** With multiple hop communication, not all disconnected sensors need to move to the vicinity of the base station to get connected. We use a *lazy movement* strategy to reduce the unnecessary movement as follows. At the end of each step, a sensor checks its neighbors to see if there are any ahead of it (i.e., closer to its current destination); if so, then it chooses the nearest neighbor as its candidate *path parent*. A sensor with a path parent can stop moving for a period, in the hope that the path parent can become connected, thereby saving its own movement. A sensor can take a neighbor as a real path parent, only when that neighbor is not adopting the sensor itself as a path parent. To avoid deadlock, if a sensor has not moved for a certain number of periods, it sends a `PathParentInquiry` message once a period to the path parent, which forwards the message to its own path parent if available, and so on. If the sensor finally receives such a message sent by itself, then it knows that a loop has been formed, and it simply resumes its suspended walk at the beginning of the next step; the disregarded path parent will not be taken as its path parent anymore. A sensor stops moving only when it enters the communication range of a connected sensor, at which time it takes the connected sensor as its *parent* and changes its own state to connected.

## 4. The connectivity-preserved virtual force (CPVF) scheme

### 4.1. Achieving connectivity

Initially, all sensors are required to decide their states regarding connectivity. The sensors in the immediate neighborhood of the base station know that they are connected. Those sensors then each flood a message to the network, and sensors receiving such a message, becoming aware that they are connected too, further flood the message into the network (each sensor sends the message only once). After a certain period of time (say, 5 seconds), if a sensor still has not received such a message, it can decide that it is disconnected, and it starts to move using the BUG2 algorithm with the lazy movement strategy toward the base station.

### 4.2. Maximizing sensing coverage using virtual forces

All sensors, upon becoming connected, start to move under the drive of virtual forces (VF) in order to maximize sensing coverage. Our goal is to preserve the network connectivity throughout this stage.

The use of virtual forces in our protocol is essentially the same as in previous work (e.g., [13]): the obstacles and neighboring sensors exert repulsive forces onto a sensor, and the sum of all forces determines the subsequent direction of that sensor. In fact, the VF method is used in our scheme only for determining moving directions; the selection of step size needs special care in the interest of network

connectivity and maximum coverage: on the one hand, aggressive step sizes may push sensors beyond of communication range and thus cause network partitions; on the other hand, moving too conservatively may result in sensors being distributed far from evenly, which usually means poor coverage. Considering this, we need a sensor to be able to determine the maximum step size it can make without disconnecting the network, or the maximum *valid* step size, at the end of each step.

Suppose at time  $t$ , the beginning of some period, a sensor  $s$  needs to check whether a planned move will disconnect itself from a current neighbor  $s'$ , or the validity of that move for  $s'$ . Since the network is an asynchronous system, sensor  $s'$  may be in the middle of its own period, which we assume ends at time  $t' \leq t + T$ . Sensor  $s$  first obtains the information of  $s'$ 's current moving direction, moving speed and period ending time  $t'$  by communication. We claim that if the planned move satisfies the following two conditions, termed the *connectivity preserving conditions*, then that move will not *by itself* break the connection between  $s$  and  $s'$  during  $[t, t + T]$ :

1. the distance between  $s$  and  $s'$  at time  $t'$  is no greater than  $r_c$ ; and
2. the distance between  $s'$ 's position at  $t'$  and  $s$ 's position at  $t + T$  is no greater than  $r_c$ .

In fact, the first condition can guarantee the connection being maintained throughout  $[t, t']$  (see the proof in our full version paper). We consider  $[t, t + T]$  and emphasize “by itself” because during  $[t', t + T]$ ,  $s$  cannot (and need not) control  $s'$ 's motion, for  $s'$  may choose to abandon their connection in order to join a new (parent) sensor; if  $s'$  does wish to keep the connection with  $s$ , then it is  $s'$ 's responsibility to find an appropriate step size following the above conditions. However,  $s$  still needs to guarantee that the connection remains in case  $s'$  does not move in its next period; this is why the second condition needs to be satisfied.

With the validity criterion for a step size, a sensor can approximately determine the maximum valid step size by checking a set of possible values, for example,  $VT, 0.9 \times VT, \dots, 0.1 \times VT, 0$ , for a certain connection. Nevertheless, there remains an issue of which connections it needs to maintain. The simplest strategy for a sensor is to preserve its connections to all its current parent and children all the time. However, our experimental results show that this strategy is too conservative to produce good coverage; allowing sensors to change parent connections provides more freedom for sensors to move around and thus more chances for the uncovered areas to be explored.

To allow a sensor to connect to a new parent, care needs to be taken not to create loops in the tree. Suppose sensor  $s$  wishes to connect to a new parent  $p$ , it first needs to “lock” the tree rooted at itself so that it can check whether

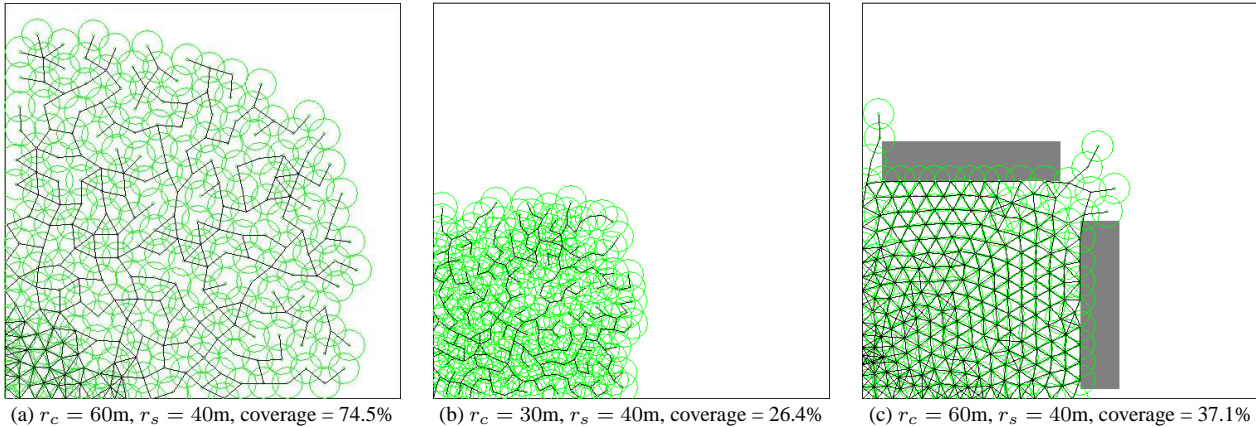


Figure 1: Sensor layouts under CPVF for varying  $r_c$  and  $r_s$ . The green circles represent sensors’ sensing ranges. The  $1000 \times 1000m^2$  field is obstacle-free in (a) and (b), and has two obstacles in (c), shown as grey areas. There are 240 sensors that can move at a speed of no greater than  $2m/s$ , with a period length of one second.

it is joining one of its descendants. It sends a `LockTree` request down the tree. If a receiving node is moving or has just decided not to change parent in its next step, then it *accepts* the request and forwards the request down the tree, or sends back the same message up the tree in the case of a leaf node; at the same time it is “locked”, meaning that it cannot change parent unless it receives an `UnLockTree` message. In other cases the receiving node *rejects* the request and sends an `UnLockTree` message back. The `UnLockTree` message travels up the tree, unlocking all the nodes on the way, until it reaches  $s$ , the request initiator. If  $s$  receives the `LockTree` message sent by itself from all its children, it knows that the tree has been successfully locked and it can further pursue joining  $p$ , otherwise it gives up and waits for a period. When joining  $p$ ,  $s$  first asks  $p$  if  $p$  is locked by  $s$ ; if not then it begins to join  $p$ , otherwise it gives up. After joining  $p$ ,  $s$  sends a `UnLockTree` message down the tree to unlock all the nodes.

There is a time limit for a sensor to determine its next step. If for any reason the above procedure takes longer than that time limit,  $s$  simply cancels its movement for the next period. Considering the message overheads of locking trees, we allow a sensor to change parent only when it cannot move sticking to its current parent.

### 4.3. Coverage performance of CPVF

We have developed an event-based simulator using C++ to evaluate the coverage performance of CPVF. In the experiments, a total of 240 sensors are initially randomly distributed in a sub-area  $\{(x, y) : 0 \leq x \leq 500m, 0 \leq y \leq 500m\}$  of a target field  $\{(x, y) : 0 \leq x \leq 1000m, 0 \leq y \leq 1000m\}$ ; the base station is located at  $(0, 0)$ . Sensors have  $r_c$  and  $r_s$  varying between  $30-60m$ ; they first move to gain connectivity, and then adjust their positions to maximize network coverage. The maximum moving speed is

$2m/s$ , and the period length is 1 second; the simulation runs for 750 seconds, after which the sensor layout becomes stable.

We define the metric *coverage* as the fraction of areas that is covered by at least one sensor. In the first scenario (Figure 1(a)), sensors have  $r_c = 60m$  and  $r_s = 40m$  and move in an obstacle-free field. It can be seen that CPVF distributes the sensors quite evenly, producing a coverage of  $78.8\%$ . When  $r_c$  reduces to  $30m$  (Figure 1(b)), the situation is very different: the smaller  $r_c$  makes the sensors cluster together, producing a much smaller network coverage of  $28.7\%$ . We can see that significant overlap of sensing disks (depicted by the circles) exists in the network, yet most sensors are unaware of this due to their poor communication ability to contact the sensors in its vicinity. Although the connected network provides the possibility for a sensor to find sensors whose sensing disks overlap with its own sensing disk, the geography-based search is not a local task and requires extra support from the protocol; and even if they can manage to make contact, significant coordination among many sensors may be needed to achieve an agreement of movement that leads to a good layout. The impact of poor coordination is further demonstrated by the poor coverage ( $37.8\%$ ) in Figure 1(c), where  $r_c = 60m$ , the same as in Figure 1(a), but two rectangular obstacles are now present in the field, leaving three exits to the big vacant area. It turns out that the sensors have great difficulty circumventing the obstacles: while a few can make their way out of the two top exits, no sensors can escape from the narrower one at the bottom of the field. Increasing the run time does not help the sensors disperse: many of them only oscillate infinitely around their centers without being able to expand coverage. This is because every sensor makes movement decisions based only on the information

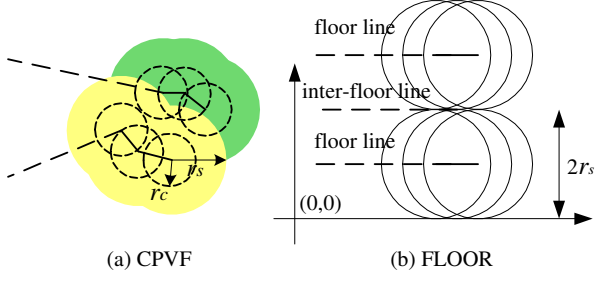


Figure 2: Examples of sensor layout under CPVF and FLOOR. The big circles represent sensing ranges, and the small circles represent communication ranges;  $r_c/r_s = 0.5$ .

of its neighbors; the localized view means that the sensors get “trapped” very easily.

## 5. The floor-based scheme

In the CPVF method, the sensors move in a greedy fashion to establish connectivity; this can result in arbitrary overlaps of the sensors’ sensing disks. When  $r_c/r_s$  is small, the sensors lack information that can guide them to move apart in such a way that the coverage is maximized. A common case is illustrated in Figure 2(a), where the big circles represent sensing disks, and the small circles represent communication ranges;  $r_c/r_s$  is assumed to be 0.5. It can be seen that despite the significant overlap of sensing disks between the two groups (green and yellow) of sensors, these sensors are unaware of this because of their short communication ranges.

The key idea of our floor-based method is to divide the field into *floors* of common height  $2r_s$ , and make sensors try to stay in the central lines, called *floor lines*, of the floors, as shown in Figure 2(b). Now that sensors are separated by floors, the overlap of the sensing disks is much reduced, and consequently the global network coverage can be improved.

The floor-based scheme has three phases: achieving connectivity, identifying movable sensors, and expanding coverage. We describe these in turn in the following sections.

### 5.1. Achieving connectivity

In FLOOR, a sensor does not move toward the base station straightway in order to establish connectivity, as is the case in CPVF. Instead it needs to pass through two intermediate destinations. Suppose the initial location of the sensor is  $(x, y)$ , then the two intermediate destinations are  $Dest1 = (x, FloorLine(y))$  and  $Dest2 = (0, FloorLine(y))$ , where  $FloorLine(y)$  is the  $y$  coordinate of the sensor’s nearest floor line. It first sets  $Start = (x, y)$  and  $Target = Dest1$  and moves using the BUG2 algorithm. Upon reaching the floor line or hitting some obstacle, it sets  $Start = Dest1$  and  $Target = Dest2$  and moves to the second intermediate destination. When it reaches  $Dest2$  or hits an object again, it sets  $Start =$

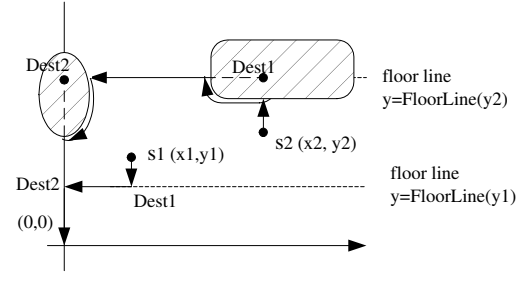


Figure 3: Sensors’ moving paths toward  $(0, 0)$  under FLOOR.

$Dest2$  and  $Target = (0, 0)$  and moves to the base station. During the course of moving, the lazy movement strategy is also used, and the sensor stops once it becomes connected. Note that in FLOOR, the  $y$  axis is regarded as a wall-like obstacle.

In Figure 3, sensor  $s_1$  moves to  $(0, 0)$  after changing two directions and without encountering any obstacles, while sensor  $s_2$  meets two obstacles preventing it reaching the exact locations of  $Dest1$  and  $Dest2$ , but following the above procedure, it finally makes its way to  $(0, 0)$ .

In FLOOR, the distance between a sensor and its parent is  $\min(r_c, 2r_s)$ . This is for the convenience of coverage hole filling, as will be introduced later.

### 5.2. Identifying movable sensors

After a sensor becomes connected, it reports to the base station, and waits for a response from the base station. The response message will reach the sensor carrying the IDs of all the sensor’s ancestors, which will be kept in the sensor’s memory. The base station initiates the second phase after all or most (say 95%) sensors have reported arrival, or a certain time (e.g., an estimate of the maximum time for a sensor to arrive at the base station) has elapsed, whichever is earlier.

The purpose of the second phase is to find out sensors that can move without partitioning the network and whose move is expected to bring a gain in coverage. The former condition requires a sensor to help all of its children to find new parents. This can be done as follows. It first obtains a list of neighbors within two hops of itself, and then tries to find for each child a new parent. Now for a particular child, it needs to check if the child joining some candidate new parent will cause a loop, which can be done by simply checking the candidate new parent’s ancestor list. If all the children can find parents to join without creating loops, then it means that the sensor can safely move away. To become a *movable* sensor, such a sensor needs to satisfy another condition: improving the network coverage. The sensor can use a simple heuristic to decide whether its move is worthwhile: it calculates the area currently covered exclusively by itself; if such an area is beyond  $\lambda A_{max}$ , then it does not move. Here  $\lambda$  is a parameter (say 60%), and  $A_{max}$  is the

maximum area that can be exclusively covered by itself in a connected network, that is, the area of its sensing disk minus its sensing overlap with another sensor at a distance of  $r_c$ .

If the above two conditions are satisfied, a sensor marks itself to be *movable*, otherwise it is termed a *fixed* node. It can then ask the children to establish connections directly to their new parent and then update the ancestor list accordingly. A movable node will not be considered as the parent of other nodes. To ensure state consistence, only one sensor is allowed to perform the above procedure at any one time. This can be coordinated by a message traversing the tree in, for example, a depth-first manner. Such a message can be initiated by the base station at the start of this phase. Upon receiving such a message, a sensor node can start to decide whether it is movable or fixed. After that, it forwards the message to others. To make sure the tree is traversed properly while the tree is being modified, every sensor needs to maintain a copy of pointers to all parent and children in the original tree.

### 5.3. Determining the coverage status of a point

A sensor often needs to determine whether a point in its vicinity is covered by others, so that it can decide whether it needs to invite some movable sensor to fill in that uncovered area. As demonstrated before, local communication is inherently limited in its ability to detect the coverage status of a point beyond a sensor's sensing range, especially when  $r_c/r_s$  is small; thus some kind of non-local communication is inevitable.

With the field being divided into floors, we are able to implement a simple scheme that helps a sensor determine the coverage status of a certain point. We define a *floor header node* as the node with the smallest absolute value of x-coordinate in a floor. (Ties can be broken by nodes' IDs.) Each floor header node maintains a data structure that records the locations of the nodes in its floor. Since many fixed nodes are linked with the same (or approximately the same) inter-node intervals and share the same y-coordinate (i.e., staying at the same floor line), their locations can be summarized in a compact form: given a sequence of such nodes, only the first and the last nodes' x-coordinates need to be remembered. When a node  $s$  needs to determine whether a point  $p$  beyond its sensing range is already covered, it firsts checks whether any of its neighbors covers that point; if not, it then calculates the possible floors that may contain a node whose sensing range can reach  $p$ . If  $p$  is located right on a floor line, then the only such floor will be  $p$ 's current floor, since the floor height is  $2r_s$ ; otherwise there will be at most two floors that may contain a node whose sensing range covers  $p$ .  $s$  then sends a query message to the floor header nodes corresponding to those floors. Those floor header nodes can determine whether  $p$  is covered by any sensor in their own floors, and send back

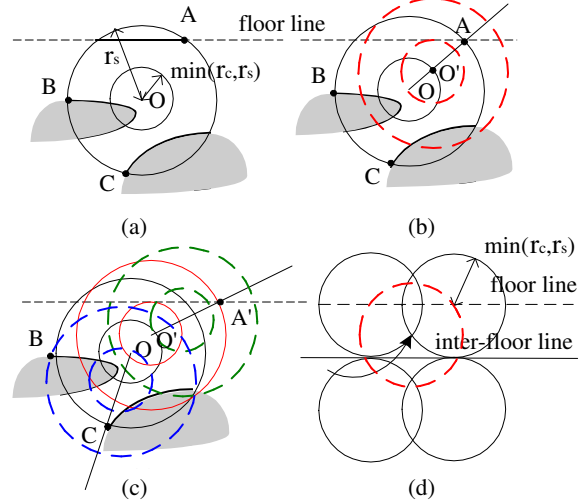


Figure 4: Coverage expansion. (a, b, c) Floor/boundary line guided expansion, where the inner circle of radius  $\min(r_c, r_s)$  represents the expansion circle; (d) inter-floor-line guided expansion. The grey irregular areas represent obstacles.

responses, from which  $s$  can determine  $p$ 's coverage status in the field.

### 5.4. Expanding sensing coverage

With all movable sensors identified, the network can now expand its coverage by moving those sensors to the uncovered areas. We identify two types of operations for expanding the coverage, namely the *floor/boundary line guided expansion* and *inter-floor line guided expansion*. For each type of operation, a sensor needs first to find an *expansion point* (EP) on its *expansion circle*, defined as the circle of radius  $\min(r_c, r_s)$  centered at its current location, and then to invite some movable sensor to relocate to the EP.

#### 5.4.1 Floor/Boundary line guided expansion

A sensor  $s$  first obtains a set of floor/boundary line segments covered by its sensing disk (the  $y$  axis is also treated as a boundary). Then, it randomly picks a line/curve segment, say  $\Gamma$ , and chooses a random point on  $\Gamma$  and makes that point "move" along  $\Gamma$  following a left-hand rule, until the point reaches the sensing circle; the stopping point is called a *frontier point* for  $\Gamma$ . Notice that the left-hand rule is opposed to the right-hand rule used by the path planning algorithm, which means that the coverage expansion is performed in the reverse direction of the sensors' assembling path. This can help sensors move into unexplored area most effectively.

Next,  $s$  determines whether that frontier point is covered by other sensors using the method introduced in Section 5.3. If  $s$  finds that the frontier point is already covered, then  $s$

gives it up; otherwise  $s$  calculates the intersection between its expansion circle and the line segment from  $s$ 's current location to the frontier point; the intersection point is the EP for  $\Gamma$ . Now  $s$  can start inviting a movable sensor to move to that intersection point. After that movable sensor has agreed to move, the above process is repeated.

Figure 4 shows an example of the above process. In Figure 4(a), the sensor centered at  $O$  detects three line/curve segments (the thick lines in the figure) in its sensing disk: one from the floor line, and two from the two obstacles. Using the left-hand rule, it can obtain three frontier points,  $A$ ,  $B$ , and  $C$ . Suppose  $s$  has just found out that  $A$  is not covered by others, it then takes  $O'$ , the intersection between its expansion circle (the inner circle in the figure) and the line segment  $OA$  as an EP. For  $O'$ ,  $s$  can invite a movable sensor to relocate to it, resulting in the layout as shown in Figure 4(b). Next,  $s$  can continue to find movable sensors to relocate to  $B$  and  $C$ . At the same time, the sensor centered at  $O'$  can determine its own EP and invite sensors to move to that EP. A possible result is shown in Figure 4(c).

While EPs on the floor lines are the major approaches for individual floors to be covered, EPs on the boundary lines are important for introducing sensors to empty floors, which cannot by themselves invite external sensors.

#### 5.4.2 Inter-floor line guided expansion

This type of expansion is used for filling the coverage holes left by neighboring sensors in the same floor. A sensor and its child can jointly determine if there is a coverage hole between themselves and a inter-floor line, which is defined as the line in the middle of two neighboring floor lines. If a hole is found, then the parent uses checks if the intersection point of their expansion circle at the side of the hole is already covered by others. If not, then the intersection point is taken as an EP, and the parent sensor can invite a movable sensor to move over; see an example in Figure 4(d). Usually there are two holes for a parent-child sensor pair, so the parent sensor needs to perform the expansion twice.

#### 5.4.3 Inviting movable sensors

The sensors check the sensing coverage and obstacles in their surrounding areas once a period and see whether there is any chance for expansion. If a sensor finds no expansion points on its expansion circle after a certain number of trials, then it stops the checking process. Otherwise, it periodically (once a period) sends an Invitation message to the network. This message has a TTL value and walks randomly in the network. A movable sensor keeps a copy of each invitation message passing through it, and when it collects a certain number of such messages, or after a certain time has elapsed, it decides whether or not to relocate. First, it picks the invitation messages according

to the decreasing priority order: floor-line guided expansion<sup>7</sup>, boundary-line guided expansion, and inter-floor-line guided expansion, and then selects the message with the nearest source (i.e., the inviter) in terms of Euclidean distance (which is only an approximate estimate of the real distance when obstacles are present). It sends an Accept-Invitation message to that inviter, which acknowledges the message if the inviter has not found any other movable sensor, or rejects it otherwise. In the former case, an agreement has been reached between the two sensors and thus the movable sensor can start moving, at the same time the inviting sensor constructs a virtual node in the tree representing the invited sensor, and also sends a message to the root on behalf of the invited sensor to update the location information maintained by its ancestors; in the latter case, the movable sensor simply continues to collect Invitation messages and responds to another inviter when appropriate. Once a movable sensor moves to an EP, it stabilizes and can start looking for expansion chances around itself.

## 6. Performance evaluation

In this section we conduct more comprehensive performance evaluation for the proposed schemes. Besides coverage, we also consider the moving distance of sensors, which dominates energy consumption in the deployment process. The same settings of field, sensor movement pattern, as well as the ranges of  $r_c$  and  $r_s$  as in Section 4.3 are used. We consider two types of initial sensor distributions: uniformly random distribution, and highly clustered distribution, in which sensors are randomly distributed in a sub-area  $\{(x, y) : 0 \leq x \leq 500, 0 \leq y \leq 500\}$ . The first distribution is used to test how the sensors form a connected network from a possibly very sparse initial distribution; while the second distribution is intended to test how sensors move apart from a clustered distribution to maximize network coverage. Due to space reasons we only report the results for the second distribution.

### 6.1. Coverage in obstacle-free fields

Figure 5 shows the coverage of CPVF and FLOOR for varying numbers of sensors in an obstacle-free field. It can be seen that FLOOR outperforms CPVF in all cases, especially for a moderate or small  $r_c/r_s$ . For instance, for 240 sensors with  $r_c = 40\text{m}$  and  $r_s = 60\text{m}$ , FLOOR produces a coverage that is 110% higher than that of CPVF.

To see how FLOOR compares with an optimal coverage scheme, we implement a centralized scheme that organizes the sensors into a network following the deployment patterns proposed in [1]. The deployment patterns have a provably optimal coverage and also achieve connectivity (but only suitable for a non-obstacle environment). Figure 5 plots the results of this optimal scheme. As can be seen, FLOOR is only surpassed by the optimal scheme by

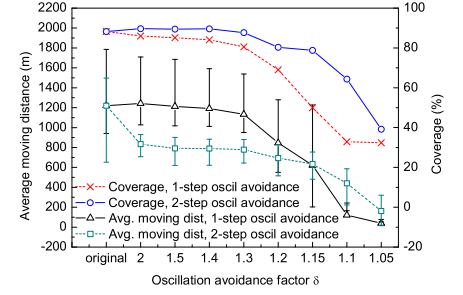
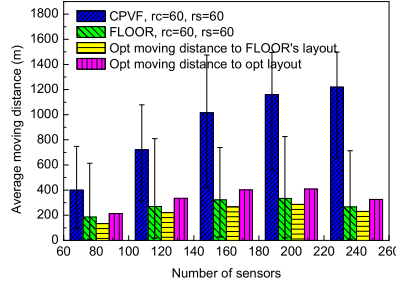
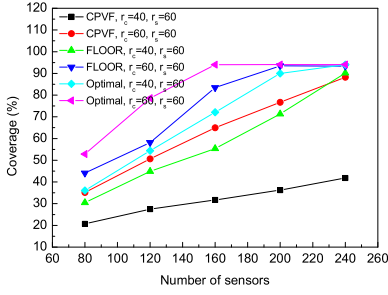


Figure 5: Coverage of CPVF and FLOOR.

Figure 6: Average moving distance.

Figure 7: Effect of oscillation avoidance on moving distance (left y-axis) and coverage (right y-axis).

a small to moderate margin, with the largest difference occurring at  $r_c = r_s = 60$  with 120 sensors, where the coverage of FLOOR is 26% lower than the optimal. As  $r_c$  and the number of sensors grows, FLOOR's coverage becomes very close to the optimal. For instance, for  $r_c = r_s = 60$  with more than 200 sensors, FLOOR's coverage is almost the same as the optimal.

The difference between FLOOR and the optimal is mainly due to some sensors that are located at the boundary of the field, where the coverage holes are smaller than those found at the interior of the field, and therefore the coverage gains are smaller than average. In a system where a central coordinator is absent, achieving the perfect (and strict) pattern will need more sensor relocations, which leads to more energy consumption. It is FLOOR's choice to trade a small amount of coverage for a saving in energy consumption. This point will be further demonstrated in the next section.

## 6.2. Moving distance in obstacle-free fields

We compare the moving distance of sensors under different schemes. Besides CPVF and FLOOR, we also implement two optimal schemes. We let a sensor move to its destination position in the final network layout, which we assume is known in advance, in a straight line. Choosing for each sensor a destination position, therefore, can be modelled as a classic *minimum weighted bipartite matching problem* (or an *assignment problem*), where the objective is to find the assignment that causes the smallest total moving distance. In the first optimal scheme, we let the final layout be the optimal pattern proposed in [1], and in the second, we set the final layout to be the one produced by FLOOR itself. The first optimal scheme can show the minimum moving distance for achieving an optimal coverage, while the second scheme shows how far FLOOR is from the lower bound of moving distance for achieving its own coverage. We use the well known *Hungarian algorithm* [3] to

solve the matching problem.

From Figure 6, one can see that the cost of FLOOR is lower than the moving distance required by the optimal pattern. This is so because when planning node movement, FLOOR takes a more comprehensive consideration of coverage and moving distance – if a sensor finds that its move brings only a small coverage gain, then the sensor stays as a fixed node. Although sacrificing a certain amount of coverage, this strategy can effectively reduce the moving distance. In contrast, the optimal pattern only considers total coverage, thus potentially involves more node movement. These results again demonstrate FLOOR's choice in striking a balance between maximum coverage and minimum energy consumption.

It can also be observed that FLOOR has a moving distance 15.6-38% higher than the lower bound for achieving its own coverage. This is no surprise since in FLOOR, the matching between coverage holes and movable sensors is performed in a distributed and greedy manner.

Figure 6 also compares the average moving distance (along with 80% confidence intervals) of the CPVF and FLOOR. It can be seen that CPVF generally requires more than two times the average moving distance of FLOOR. The reason behind this is that the oscillation happens very often in CPVF, that is, many sensors move back and forth under the drive of virtual forces, making a lot of unnecessary moves on the way to their destinations.

We consider two oscillation avoidance techniques in an attempt to reduce unnecessary movement in CPVF. In the first technique, called *one-step oscillation avoidance*, a sensor cancels its movement for the next step if it finds that the next step size is smaller than  $VT/\delta$ , where  $VT$  is the maximum step size and  $\delta$  is a parameter called *oscillation avoidance factor*. This is intended to eliminate small perturbations and force the system to enter an equilibrium state earlier; see [5] for a similar strategy. In the second technique, called *two-step oscillation avoidance*, a sensor com-

compares its future location at the end of the next step with its past location at the end of last step; if the distance between them is smaller than  $VT/\delta$ , then it cancels its movement plan for the next step. A similar approach has been used in [9]. Figure 7 shows the impact of varying the  $\delta$  value on the moving distance (the left y-axis) and coverage (the right y-axis) of both schemes. It can be observed that  $\delta$  does have the effect of reducing moving distance; however, this comes at the cost of reduced coverage. When  $\delta$  gets smaller, it is more likely that a step will be regarded as an unnecessary or repeated move, thus more steps are cancelled. On the other hand, the conservative moving of sensors prevents them from exploring the uncovered areas effectively, resulting in poor coverage. We have also tried oscillation avoidance based on longer movement history, and found similar trends. The difficulty of avoiding oscillation without reducing coverage is that some seemingly unnecessary steps are actually effective moves that can potentially push the frontier of coverage forward, yet it is very hard to tell which of those moves are effective, so that we can know when to start the oscillation avoidance and when to stop it. The failure of reducing oscillations by local decisions again reflects the limitation of the VF-based approaches.

### 6.3. Fields with random obstacles

We next consider sensor deployment in a field with obstacles. The number of obstacles are randomly drawn from  $\{1, 2, 3, 4\}$ . Possibly overlapping with one another, the obstacles are rectangular and of random sizes, under the condition that they do not partition the field. The number of sensors is fixed to 160. A total of 300 runs of the simulation are executed, and the coverage and moving distance results of the two schemes are recorded. Figure 8 depicts the cumulative distribution function of coverage and average distance. In this general obstacle setting, it can be seen that FLOOR outperforms CPVF significantly in terms of both coverage and moving distance. For instance, while the mean coverage of FLOOR is more than 20% higher than that of CPVF, the mean moving distance is less than half of CPVF.

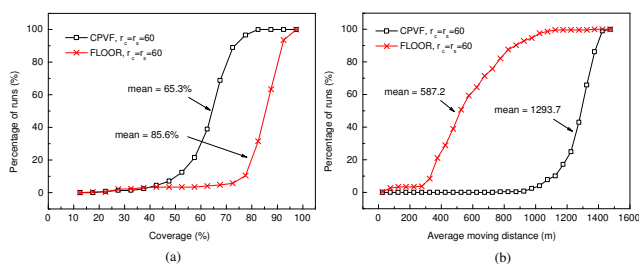


Figure 8: CDFs of (a) coverage and (b) distance under random obstacles.

## 7. Conclusion

We have presented two sensor deployment schemes for mobile sensor networks in general 2-D fields. The major difference of our schemes from prior work is their adaptiveness to arbitrary network densities or communication ranges, and to obstacles. Our first scheme is an enhanced form of the traditional virtual force-based method, which is shown to perform well only in very restricted scenarios. We then propose a floor-based scheme which shows good performance in all cases.

## References

- [1] X. Bai, S. Kumary, D. Xuan, Z. Yun, and T. H. Lai. Deploying Wireless Sensors to Achieve Both Coverage and Connectivity. *Proc. of MobiHoc 2006*.
- [2] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Trans. on Robotics and Automation* 20(2), 243-255, 2004.
- [3] Hungarian Algorithm. [http://en.wikipedia.org/wiki/Hungarian\\_algorithm](http://en.wikipedia.org/wiki/Hungarian_algorithm)
- [4] N. Heo and P. K. Varshney. Energy-Efficient Deployment of Intelligent Mobile Sensor Networks. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 35(1):78-92, Jan. 2005.
- [5] 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 *Proc. of the 6th International Symposium on Distributed Autonomous Robotics Systems (DARS02)*, 2002.
- [6] Y. Koren and J. Borenstein. Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. *Proc. of the IEEE Conference on Robotics and Automation*. California, USA. 1991.
- [7] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403-430, 1987.
- [8] G. Wang, G. Cao, T. L. Porta, and W. Zhang. Sensor Relocation in Mobile Sensor Networks. *IEEE INFOCOM 2005*.
- [9] G. Wang, G. Cao, and T. L. Porta. Movement-Assisted Sensor Deployment. *IEEE INFOCOM 2004*.
- [10] G. Wang, G. Cao, and T. L. Porta. A Bidding Protocol for Sensor Deployment. *IEEE ICNP 2003*.
- [11] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill. Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks. *ACM SenSys 2003*.
- [12] S. Yang, M. Li, and J. Wu. Scan-Based Movement-Assisted Sensor Deployment Methods in Wireless Sensor Networks. *IEEE Trans. on Parallel and Distributed Systems*, 2006.
- [13] Y. Zou and K. Chakrabarty. Sensor Deployment and Target Localization Based on Virtual Forces. *INFOCOM 2003*.