Uniform Dispersal of Asynchronous Finite-State Mobile Robots in Presence of Holes *

Eduardo Mesa-Barrameda[1]   Shantanu Das[2]   Nicola Santoro[1]

[1] Carleton University, Canada, *{santoro}@scs.carleton.ca*
[2] LIF, Aix-Marseille University, France, *shantanu.das@lif.univ-mrs.fr*

Abstract

We consider the problem of uniformly dispersing mobile robots in an unknown connected orthogonal space. The robots are autonomous and identical, they enter the space from a single point, and move in coordination with other robots, relying only on sensed local information within a restricted radius. The existing solutions for the problem require either the robots to be synchronous or the space to be without holes and obstacles. In this paper we allow the robots to be fully asynchronous and the space to contain holes. We show how, even in this case, the robots can uniformly fill the unknown space, avoiding any collisions, when endowed with only $O(1)$ bits of persistent memory and $O(1)$ visibility radius. Our protocols are asymptotically optimal in terms of visibility and memory requirements, and these results can be achieved without any direct means of communication among the robots.

1 Introduction

Unlike their static counterparts, mobile sensors and robots can self-deploy within a target space S to "cover" it so to satisfy some optimization criteria. To achieve such a goal without the help of any central coordination or external control is a rather complex task, and designing localized algorithms for efficient and effective deployment of these mobile entities is a challenging research issue. Such a task has been studied by several authors and continues to be the subject of extensive research. Most of the work is focused on the (uniform) *self-deployment* problem; that is, how to achieve uniform deployment in S (usually assumed to be polygonal) starting from an initial random placement of the sensors in S (e.g., [2, 6, 7, 8, 9, 10, 12, 13, 16, 17]).

The problem has recently been studied within the context of weak robots: the mobile entities rely only on sensed local information within a restricted range, called *visibility radius*; usually they have no explicit means of communication or with a very limited radius, called called *communication range*. Localized solution algorithms for such weak robots have been developed for special spaces such as a *line* (e.g., a rectilinear corridor), a *ring* (e.g., the boundary of a convex region), a *grid* , etc. (e.g., [1, 3, 4]; see [5] for a recent survey).

We are interested in a specific instance of the self-deployment problem, called the *Uniform Dispersal* (or *Filling*) problem, where the robots have to completely cover an unknown space S entering through a designated entry point. In the process, the robots must avoid colliding with

*Research partially supported by NSERC Canada.

each other, and must terminate (i.e., reach a quiescent state) within finite time [9, 11, 15]. The space S is assumed to be orthogonal, i.e. polygonal with sides either parallel or perpendicular to one another; orthogonal spaces are of particular interest because they can be used to model indoor and urban environment (e.g., floorplans, city maps, etc).

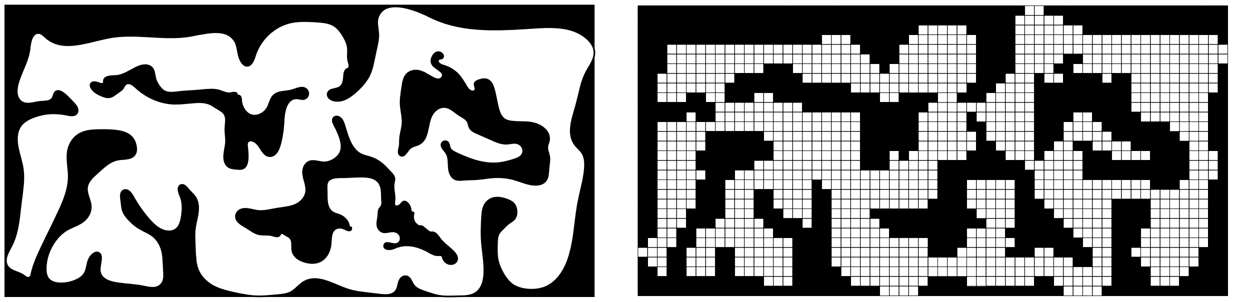
Figure 1: An arbitrary closed space (Left) represented as an orthogonal grid of cells(Right). Each cell can contain a single robot.

Our focus is on the minimum capabilities required by the sensors in order to effectively complete the uniform dispersal task.

There are some intrinsic limitations on the type of memory and the visibility/communication radius needed to solve the uniform dispersal problem for any connected orthogonal space whose shape is a priori unknown.

In particular, it is known that the robots need to have some *persistent* memory of the past; the problem is in fact unsolvable by *oblivious* robots even if the system is fully synchronous and the robots have unlimited visibility and unlimited communication radius [15]. It is similarly unsolvable by robots that cannot see nor communicate, even if the system is fully synchronous and the robots have unlimited persistent memory [15].

Since oblivious robots cannot deterministically solve the problem, the question is whether the problem is solvable by *finite-state* robots, that is robots with a constant number of bits of non-volatile memory. More specifically, since robots that cannot see nor communicate cannot deterministically solve the problem, the main research question is whether the problem is solvable by finite-state robots that have just a constant visibility radius (and constant communication range) .

Two existing results hints that, under very restrictive conditions, the answer to those questions is positive. In fact, it has been proven that finite-state robots with constant visibility and communication radius are able to solve the problem if the system is *fully-synchronous* (FSYNC), which allows perfect coordination between the robots [11]. A similar positive result has been established for *asynchronous* systems (ASYNC) if the space S contains *no holes* (i.e., obstacles completely within S) [15].

In this paper we lift these two restrictions: we allow the robots to be fully asynchronous and the space to contain holes; and we the show that the answer is indeed positive in all cases. We constructively prove that robots endowed with only $O(1)$ bits of persistent memory and $O(1)$ visibility radius can always uniformly fill the unknown space, avoiding any collisions; this results can be achieved without any direct means of communication.

In particular, we present a solution protocol that, without using any direct communication, solves the problem for robots having only a few states and a visibility radius $v = 6$. We then investigate the use of direct communication of constant range and show how to use it to decrease the visibility radius, without increasing the memory requirement. Namely, we introduce a second protocol that, using communication radius $c = 1$, solves the problem for finite-state robots having visibility radius $v = 1$; transmitted messages in this protocol are of constant size. A summary of these results is shown in table 1. Due to the space constraint, some of the proofs have been omitted and they can be found in the appendix.

Algorithm	Memory of Robots	Visibility Radius	Communication Radius	Synchronous
MUTE	O(1)	6	no communication	NO
TALK	O(1)	1	1	NO

Table 1: Summary of contributions.

2 Model and Definitions

The system is composed by a set R of mobile entities, called *robots*, whose task is to completely cover a space S that they enter sequentially from the same place.

The space S is a connected finite orthogonal region of the plane possibly with holes; the shape of S is arbitrary (see Figure 1). Connected means that it is possible to reach any point of S from any other point of S; a *hole* is a region of the plane which is not part of S and is surrounded completely by points of S; the boundaries of S and holes are called *obstacles*. The region S and its holes are assumed to be partitioned into square cells; each cell can be covered completely by a robot, and each hole is composed of an integral number of cells. Let $|S| = n$ be the number of cells of S.

The robots in R are simple computational entities with sensory and locomotion capabilities.The sensory devices on the entity allows it to have a vision of its immediate surroundings up to a fixed distance called visibility radius[1] v. The robots may have or may not have an explicit means of communicating; if available, this ability is also restricted to a fixed distance called communication range[2] c, and each transmission is restricted to a constant number of bits. Both the visibility radius and the communication range remain constant in time. The robots have local sense of orientation, that is a consistent notion of *up-down* and *right-left*; however, they do not have any global positioning mechanism, and have no knowledge of the shape of S other than that it is connected and orthogonal. The robots are *anonymous*, in the sense that there are no distinct ids and they are externally identical, autonomous, and they all follow the same protocol. Each robot has a constant number of bits of non-volatile working memory; thus the robots are *finite-state* machines with s distinct states.

On entering S, a robot operates in continuous *active-inactive* cycles. When *active*, a robot performs a *Look-Compute-Move* sequence of operations: It first takes a snapshot of its surrounding inside the visibility radius to know which cells are occupied, empty or obstacle (*Look*). Using the information provided by the snapshot and its local state, the robot executes the protocol to determine a new state and a destination cell, which is either the same where it currently resides or one of the four adjacent cells (*Compute*). Finally, it moves to the computed destination (*Move*).

[1]A visibility radius of one means that the robot sees all eight neighboring cells.

[2]A communication range of one means that the robot can communicate directly to the robots located in the eight neighboring cells.

It then becomes *inactive*. In case the robots have communication capabilities, a robot may send messages to any robot within its communication range, during the *Compute* stage. The message is immediately received and causes the receiving robot to change its state. Each cycle of activity is assumed to be non-interruptable, in the sense that once they are started they will be completed. However, the robots are *asynchronous*: there is no global synchronization among the cycles of different robots, and the time elapsed between two consecutive operations by the same robot, as well as between two consecutive activations, is finite but arbitrary.

The robots enter S through a special cell called *door*. This cell could be located anywhere and it is indistinguishable from other cells; that is, a robot cannot distinguish the door from other cells using its sensory vision. However, unlike other cells, the door is never empty: whenever a robot leaves the door cell, a new entity appears it. If a robot is in a cell, it completely *covers* it. If two or more robots are in the same cell at the same time then there is a *collision*. Two cells are called neighbors if a robot can move from one to the other in one step. The *distance* between two cells is the smallest number of steps a robot needs to move to reach one cell from the other. The *successor* of a robot r is the robot that entered S just after r and its predecessor is the robot that entered just before r.

The problem to be solved called *uniform dispersal* (or *filling*), requires that within finite time, the entire space is completely *filled*, i.e., every cell of the space is occupied by exactly one robot. Furthermore the system configuration at that time must be *quiescent*, i.e., no robot moves thereafter.

3 Algorithm for Filling without Communication

In this section, we consider robots that do not have any explicit means of communication. The robot can still see other robots (if they are within the visibility range). So the only way for robots to coordinate with each other is by means of their vision and their movements. We present an algorithm (**MUTE**) that succeeds in filling the entire space with such robots, without any collisions, even under the restriction that both the amount of memory available to a robot and its visibility range are constants (independent of the size of the space and the number of robots).

The strategy of *Follow the Leader* introduced in [11], solves the filling problem by moving the robots in a single file, with one leading robot. However, when there are holes, it is possible there are cycles in the path of the leader, which would lead to either a deadlock or a collision. To avoid forming any cycles, the algorithm **MUTE** uses the simple trick of putting up a wall to block any secondary access to the path. In other word, whenever the Leader reaches a cell from which it can move in more than one direction (henceforth such a cell is called a *bifurcation* cell), the leadership is passed to the next robot and the old leader moves to one of the neighbouring cell to block this access until the current path is completely filled.

During the algorithm **MUTE**, the robots can be in one of the following four states:

- **Leader:** There is at most one robot in this state at any time during the execution of the algorithm[3].
- **Follower:** A robot in this state is always in the path followed by the current leader.
- **Blocking:** A robot in this state was a Leader that reached a bifurcation cell and moved to block one of the access to the path. A robot in this state remains stationary until all other

[3]Except during the process of transferring the leadership as explained later

possible directions[4] become completely filled. At this moment, if it is possible for this robot to move, then it reassumes the leadership and starts moving again.

- **Stopped:** Any robot in this state was a Leader that reached a dead-end. This is a terminating state so robots that enter this state never move again.

The transition from one state to another, during any execution of algorithm **MUTE**, is defined by the diagram shown in Figure 2.

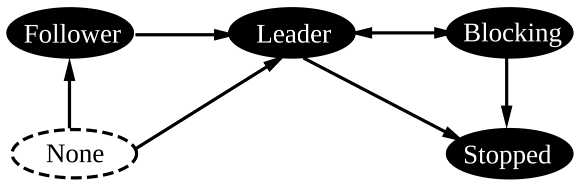

Figure 2: States transition diagram of algorithm MUTE where *None* is the state of the robot when it appears at the door.

Note that the robots need to coordinate with each other to pass the leadership, exchange state information, avoid collisions and so on. Since the robots lack any means of explicit communication, we need to use the visual capabilities of the robots to implement some sort of implicit communication. In algorithm MUTE, this is achieved by assuming a visibility range of six cells (i.e. the robots can see up to a distance of six cells in each direction). This visibility range is enough to permit each robot to see the preceding and succeeding robots, and still maintain an appropriate distance between them so that the robots can move in special patterns as a signalling mechanism to exchange information whenever necessary. In order to communicate implicitly using vision, both robots, the sender and the receiver of the information, effectuate some special movements. This is reminiscent of the way some species of animals or insects communicate among themselves (e.g. the dancing of the bees when they find a source of food).

Thus, during the algorithm the robots move along a path maintaining a fixed pattern (i.e. a fixed distance between successive robots). This pattern is broken when a robot (typically the leader) needs to communicate with its successor. The successor realizes the break in the pattern and thus, it knows that a communication process has started. The receiver acknowledges that it has received the information by making a special movement (specified later). Once the information exchange is completed, the two robots continue moving along the path, maintaining the fixed pattern as before. Algorithm 1 presents the rules for movement of a robot r when it is not is the process of communicating with another robot. Algorithm 2 defines the rules for two consecutive robots in the path that are in the process of communicating something.

During the algorithm, the system of robots is always in a *correct configuration* as defined below.

Definition 1 *During algorithm* **MUTE***, at any time t, the system is said to be in a correct configuration if and only if the following conditions hold:*

- *There is at most one Leader, or, the leadership is being passed from one robot to another (by means of a special communication process).*

[4]the blocking robot only takes into account the direction that were still open at the moment it enters in the Blocking state

Algorithm 1 (MUTE): Rules for movement

A robot r moves forward (except when it is in the process of communicating) if and only if the following conditions hold:

1. Robot r is not is state **Blocking** or **Stopped**.
2. The cell to which r wants to move is not **occupied**
3. r does not have a predecessor **or** the predecessor is at distance four.
4. r does not have a successor **or** the successor is at distance three **or** the successor is at the door, at a distance less than three.

- *Let Π be the path followed by the Leader from the door to its current location (If the leadership is being passed we consider the old leader), then:*
  - *Except the Leader, any robot located on Π is in state Follower.*
  - *Any robot in state Follower is located in Π.*
  - *Every cell $p' \notin \Pi$ that has a neighboring cell $p \in \Pi$ is occupied by a robot either in state Blocking or in state Stopped.*
- *Let $R = \{r_i : 0 \leq i \leq l\}$, be the ordered sequence of robots in Π where l is the number of robot in state Follower, r_l is the leader and r_0 is the robot at the door. And let r_i and r_j, $i, j > 0$, $j = i + 1$, be two consecutive robots in R*
  - *If neither r_i nor r_j is in the middle of a communication process then the distance in Π between them is either three or four.*
  - *If r_i and r_j are at distance less than three, then they are in the process of communicating with each-other.*

Note that the robot r_0 which is at the door, is not required to satisfy the last condition of a correct configuration. This is because, whenever the door becomes empty, a new robot appears at the door. Thus, the pattern of distances cannot be maintained near the door. However, as we will show this does not affect the behavior of the rest of the robots.

We shall now explain in more details when and how communications take place in algorithm **MUTE**. During any execution of the algorithm there are four situations in which a robot needs to communicate some information to its successor. We describe below these four types of communications called **Blocking**, **Stopping**, **Bifurcation** and **Reopening**.

1. **Blocking:** This situation occurs when the leader reaches a cell with more than one possible direction of movement (see figure 4). In this case the leader (the dark-gray robot in the figure) does not need to take any special action. The leader waits until its successor (the white robot) is at distance three; it then moves to one of the possible directions and changes its state to Blocking (the light-gray robot in the figure). When the successor robot was at distance four (figure 4 (a)), it realizes, before its next move, that its predecessor has more than one possible movement. So after the move, the successor becomes the new leader (the dark-gray robot in figure 4 (b)), while the predecessor changes to state Blocking. As a

Algorithm 2 (MUTE): Rules for implicit communication

Robot s needs to notify something to its successor r which is at distance three (see Figure 3).

1. s waits until r's successor becomes visible. This is to ensure that r's successor knows that there is a communication process in progress ahead so it does not move until the communication is completed.
2. s moves one step back to notify a communication of either a **Bifurcation** or **Reopening** (Figure 3 (b)). As explained later, the visibility range of the robots and the pattern maintained by the robots in the path, allows robot r to realize which kind of communication is taking place. After the move, robot s waits for an acknowledgement from robot r.
3. To acknowledge the receipt of the communication, robot r now moves one step ahead and waits for a notification from robot s (Figure 3 (c)).
4. s signals the end of the communication process by moving one step ahead; it then waits for an acknowledgement again (Figure 3 (d)).
5. Robot r acknowledges by moving one step back. At this time, the communication has ended and both robots are back in their original positions (Figure 3 (e)).

result, the leadership was passed from one robot to another and the communication process ends here.

2. **Stopping:** This situation occurs when the leader r_l is moving to the end of the path and it has only one more move to perform (Figure 5(b)). In this case, the Leader's successor r_s, cannot distinguish whether its predecessor is in the middle of a communication process with some robot ahead (Figure 3) or, the predecessor is the leader going to a dead-end (Figures 5 (b) and (e)). Thus r_s will wait at distance four from r_l. The Leader r_l moves one step ahead to communicate to the successor r_s that the leader has reached a dead end and the leadership should be passed to another robot. Note that if r_l was at a bifurcation cell (before the move) then it will now become **Stopped** (Figure 5 (c)). In this case, the next **Blocking** robot will notify r_s which is the new direction. Otherwise, if r_l was not at a bifurcation cell, it will become **Blocking** (Figure 5(f)), and then communicate to r_s that the path is going to a dead-end. This communication is similar to the **Reopening** communication (described below) when a new direction of movement is communicated (except that the new direction is actually a dead-end in this case, and the robot r_s will realize that).
3. **Bifurcation:** This situation occurs when a robot, either a Follower or a Leader, is in a bifurcation cell with only one possible movement (Figure 6), so the robot must communicate to its successor the information that this is at a bifurcation cell.
4. **Reopening:** A *Blocking* robot b must reopen the blocked direction because all other directions that were open before, are now completely filled (figure 5 (c) and (e)). It is possible that are other *Blocking* robots in neighboring cells, but if those directions were blocked before b became *Blocking*, then robot b has the higher priority to reopen its blocked direction.

In the two last types of communication (*Bifurcation* and *Reopening*) robots must take some

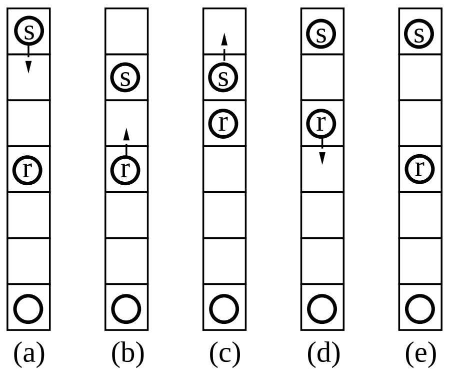

Figure 3: Behavior of sender robot s and receiver robot r, during a communication process.

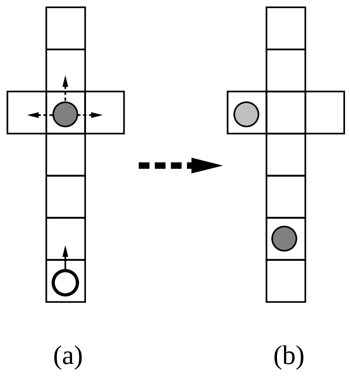

Figure 4: (a) The Leader (colored Dark-grey) reaches a cell with more than one possible moves (b) It blocks one direction and passes leadership to the successor.

special action in order to communicate. The behavior of the two robots during these two types of communication is essentially the same and follows the rules of Algorithm 2. Notice that whenever two robots start such a communication process they do nothing else until the communication has terminated. In this sense, the communication process can be thought of as an atomic action.

There are a few complications that can occur during the communication process of Algorithm 2, due to the special condition for robots at the door. First consider what happens if at the start of the communication process, the successor of the receiver (r_s) is at distance less than three from the receiver (r). There are two cases when this situation could occur. Either when r_s is at door or when r_s itself is communicating with r. In the first case r_s will not move since its predecessor is at distance less than three. And, in the second case, r_s will move back just one step returning to distance three from r. Notice that r will not move until r_s moves back since they are in a communication process. After that, r can start communicating with its predecessor s. Thus, both communications would be successful. Now, let us consider the case when the receiver robot r is at the door. In this case, since robots can see at distance six and all the robots enter the space through a unique door, any robot at the door can distinguish among the four scenarios in which some communication is needed. Thus it is not actually necessary to notify anything to a robot that is at the door. Moreover since the sender robot s would itself be within a constant distance of the door, it can remember whether its successor r is at the door or not. Thus the

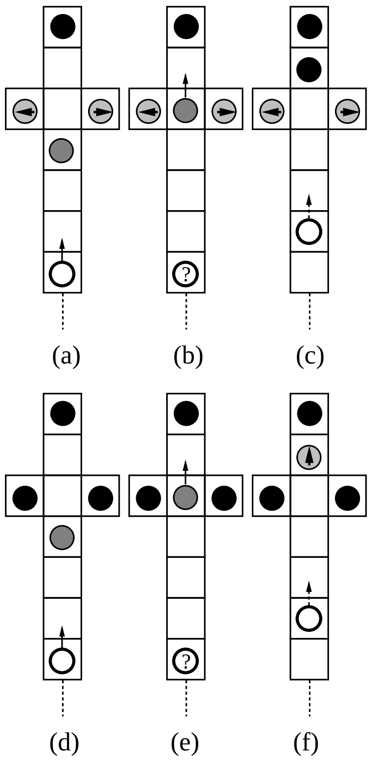

Figure 5: The Leader (colored Dark-grey) is moving to the end of the path. The white robot is a Follower which is next robot in the path. Black robots are in state Stopped, while Grey robots with an arrow are in state Blocking (i.e. blocking the exit in the direction of the arrow).

robot s would act as if the communication has already been sent to r.

We can show the following properties about the algorithm **MUTE** to prove its correctness.

Proposition 1 *The following holds during algorithm* **MUTE***: (1) Starting from a correct configuration, every communication process ends in a correct configuration. (2) Starting from a correct configuration, every movement following the rules of Algorithm 1 ends in a correct configuration.*

Proposition 2 *There are no collisions during Algorithm* **MUTE***.*

Proposition 3 *There are no deadlocks during any execution of algorithm* **MUTE***.*

Proposition 4 *Algorithm* **MUTE** *always terminates within a finite time.*

Proposition 5 *Algorithm MUTE always completely fills the space S.*

The above results prove the correctness of the algorithm. Recall that the robots can be in only five states and in each state a robot needs to remember only a constant amount of additional information. To summarize:

Theorem 1 *Algorithm* **MUTE** *solves the filling problem for any connected mesh of size n without collisions and using exactly n robots, each of which has $O(1)$ memory and a visibility radius of six.*

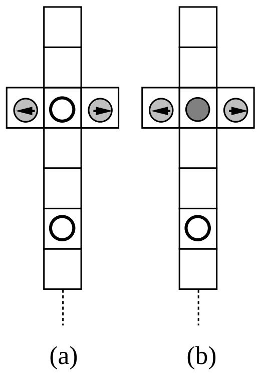

Figure 6: Either a Follower (colored White) or the Leader (colored Grey) is in a bifurcation cell.

4 Algorithm for Filling with Explicit Communication

The algorithm in the previous section used robots having a visibility radius of six to solve the Filling problem. We now show that if explicit communication is allowed between robots (even within short ranges), then the visibility radius of the robots could be reduced one. Note that this is the minimum visibility radius necessary to solve Filling without any collisions [15].

In this section we present an algorithm (**TALK**) which solves the filling problem when both the visibility and communication ranges are limited to distance one. The algorithm still requires only a constant amount of memory for each robot, and fills the complete space without any collisions. Algorithm **TALK** also uses the basic strategy of follow-the-leader, but no robot moves until its previous position is occupied by the successor. This ensures that the distance between consecutive robots in the path is never more than two (at most one cell between consecutive robots). If the leader robot r is about to move to a cell x that was visited before (e.g. when the path has a cycle), robot r can determine whether cell x has been already visited, by asking the robots in the cells neighboring x (Robot r can see at least two such cells and one of them would contain a robot). Thus the algorithm can avoid collisions. During the algorithm, the leader always moves to an unvisited cell, while the followers always move to the previous position of their predecessor. If the leader does not have any unvisited neighboring cell, it stops moving and passes the leadership to its successor.

The robots could be in one of the following states while executing this algorithm:

1. **Leader**: At any time, there is at most one robot in this state. This robot always moves to an unvisited cell. If there is no possible movement it passes the leadership to its successor and changes to state **Stopped**.

2. **Follower**: A robot in the state is always on the path from the door to the current **Leader**. A Follower robot always moves to the previous position of its predecessor unless it receive the leadership (in this case it changes to state **Leader**).

3. **Stopped**: A Robot in this state has terminated and does not ever move again during the execution of the algorithm.

A new robot appearing at the door changes to state *Follower* or *Leader* depending on whether or not it sees any robots in the neighboring cells.

The following variables and functions are used by a robot r to store and share information about past and future moves:

- *r.Entry* is the cell from which r entered its current position. This variable is NULL if r is at the door.
- *r.Exit* is the cell to which r should move in the next step. This variable is NULL for the Leader and the Stopped robots.
- *Find-Next-Move(NewCell)* is a function that returns true or false depending on whether or not there is a valid next move (i.e. an unvisited neighboring cell). The possible destination is returned as parameter *NewCell*.

The communication between robots uses the following types of messages:

1. ***L*eadership:** This message is sent by the Leader to transfer the leadership to its successor.
2. ***N*ext-Pos:** This message is sent by either a Leader or a Follower, just before moving. This informs their successor about the next position in the path.

When a robot r receives a *Next-Pos* message from the cell x, the variable *r.Exit* is set to cell x. The rules of the algorithm are presented in Algorithm 3. The implementation of function *Find-Next-Move()* is straightforward and it is omitted.

We can prove the following properties for the algorithm.

Proposition 6 *There are no collisions during Algorithm* **TALK***.*

Proof: As in the previous algorithm, a robot never moves to an occupied cell. Furthermore, according to the rules of the algorithm, no robot backtracks and a Follower robot can move only to the last position of its predecessor, when this cell becomes empty. Since only the Leader can move to unvisited cells, no collisions are possible. Therefore, algorithm TALK is collision free. ■

Proposition 7 *Algorithm* **TALK** *terminates in finite time.*

Proof: According to the rules of the algorithm, the Follower robots always move on the path of Leader while the Leader only moves to unvisited cells and passes the leadership to its successor whenever it cannot find any empty cell to move to. Since the number of cells are finite, any Leader must eventually stop and pass the leadership. Every new leader will also stop in finite time and, thus every robot on the path will stop. Therefore the algorithm terminates in a finite time when every robot has stopped. ■

Proposition 8 *Algorithm* **TALK** *completely fills the space S.*

Proof: Suppose that after termination of the algorithm, at time t, there are some cells that remain empty. Let p be such a cell. Notice that if this cell was visited before then there must be a *Follower* robot r in one of its neighboring cells and the next time this robot executes, it will move to cell p, contradicting the fact of the termination of the algorithm. Thus, every empty cell must be an unvisited cell.

However since the space is connected, there must be an empty (and hence unvisited) cell adjacent to an occupied cell q. Consider the last robot that visited cell q. This robot is now

Algorithm 3 (TALK) : Rules followed by robot r

```
if ($r.State = None$) then
  if (all neighboring cells are empty) then
    $r.State \leftarrow Leader$
  else
    $r.State \leftarrow Follower$
  end if
else if ( $r.State = Follower$ ) then
  if ( ( $r.Exit \neq$ NULL ) $\wedge$ ( $r.Entry$ is occupied )) then
    $NewPos \leftarrow r.Exit$
    $r.Exit \leftarrow$ NULL
    Send message Next-Pos to $r.Entry$.
    $r.Entry \leftarrow$ Current cell.
    Move to $NewPos$
  end if
else if ( $r.State = Leader$ ) then
  if ( $r.Entry$ is occupied ) then
    if ( Find-Next-Move($NewCell$) = true ) then
      Send message Next-Pos to $r.Entry$.
      $r.Entry \leftarrow$ Current cell.
      Move to $NewCell$
    else
      Send message Leadership to $r.Entry$.
      $r.State \leftarrow Stopped$
    end if
  end if
end if
```

Stopped (since the algorithm has terminated). But according to the rules of the algorithm, a robot cannot become *Stopped* if there is an unvisited cell adjacent. This contradiction proves that there cannot be any empty cells after the termination of the algorithm. Thus, the algorithm completely fills S. ■

We summarize the above results as follows:

Theorem 2 *Algorithm* **TALK** *solves the filling problem for any connected space of size n without collisions and using only n robots each having a constant amount of memory and a visibility and communication radius of one.*

References

[1] L. Barriere, P. Flocchini, E. Mesa Barrameda, N. Santoro. "Uniform scattering of autonomous mobile robots in a grid", *Int. J. Foundations of Computer Science* 22(3): 679-697, 2011.

[2] T.M. Cheng and A.V. Savkin. "A distributed self-deployment algorithm for the coverage of mobile wireless sensor networks". *IEEE Communications Letters*, 13 (11): 877 - 879, 2009.

[3] R. Cohen and D. Peleg. "Local algorithms for autonomous robot systems". In Proc. *13th Colloquium on Structural Information and Communication Complexity*, 29–43, 2006.

[4] P. Flocchini, G. Prencipe, and N. Santoro. "Self-deployment of mobile sensors on a ring". *Theoretical Computer Science* 402(1): 67-80, 2008.

[5] P. Flocchini, G. Prencipe, and N. Santoro. "Computing by Mobile Robotic Sensors". Chapter 21 of: S. Nikoletseas, J. Rolim (Eds.), *Theoretical Aspects of Distributed Computing in Sensor Networks*, Springer, 2011.

[6] A. Ganguli, J. Cortes, and F. Bullo. "Visibility-based multi-agent deployment in orthogonal environments". In Proceedings *American Control Conference*, 3426-3431, 2007.

[7] N. Heo and P. K. Varshney. "A distributed self spreading algorithm for mobile wireless sensor networks". In Proceedings *IEEE Wireless Communication and Networking Conference*, volume 3, 1597–1602, 2003.

[8] N. Heo and P. K. Varshney. "Energy-efficient deployment of intelligent mobile sensor networks". *IEEE Transactions on Systems, Man, and CyberNetics - Part A* 35(1):78–92, 2005.

[9] A. Howard, M.J. Mataric, and G.S. Sukahatme. "An incremental self-deployment algorithm for mobile sensor networks". *IEEE Transactions on Robotics and Automation* 13(2): 113-126, 2002.

[10] 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 *6th International Symposium on Distributed Autonomous Robotics Systems* (DARS), 299–308, 2002.

[11] T.R. Hsiang, E. Arkin, M.A. Bender, S. Fekete, and J. Mitchell. "Algorithms for rapidly dispersing robot swarms in unknown environment". In Proc. *5th Workshop on Algorithmic Foundations of Robotics* (WAFR), 77-94, 2002.

[12] X. Li, H. Frey, N. Santoro, and I. Stojmenovic "Strictly localized sensor self-deployment for optimal focused coverage". *IEEE Transactions on Mobile Computing*, 10(11): 1520-1533, 2011.

[13] L. Loo, E. Lin, M. Kam, and P. Varshney. "Cooperative multi-agent constellation formation under sensing and communication constraints". *Cooperative Control and Optimization*, 143–170, 2002.

[14] E. Martinson and D. Payton. "Lattice formation in mobile autonomous sensor arrays". In Proc. *International Workshop on Swarm Robotics* (SAB'04), 98-111, 2004

[15] E. Mesa Barrameda, S. Das, and N. Santoro. “Deployment of asynchronous robotic sensors in unknown orthogonal environments”. *4th International Workshop on Algorithmic Aspects of Wireless Sensor Networks* (ALGOSENSOR’08), 125-140, 2008.

[16] S. Poduri and G.S. Sukhatme. “Constrained coverage for mobile sensor networks”. In Proc. *IEEE Int. Conference on Robotic and Automation*, 165–173, 2004.

[17] G. Wang, G. Cao, and T. La Porta. Movement-assisted sensor deployment. In Proc. *IEEE INFOCOM*, volume 4, pages 2469–2479, 2004.

APPENDIX

Proof of Proposition 2
In algorithm **MUTE**, robots may backtrack only during a communication process. During the communication process, the sender moves back only when the receiver (its successor) is at distance three, and the rules of the communication process ensure that these two robots do not collide. Moreover these two robots do not go beyond the stretch of path between them (when they started the communication), and all cells neighboring the path are blocked, in any correct configuration. Thus no collisions can occur during a communication process.

Let us now assume that no communication takes place and suppose a collision occur the normal movement of the robots. Since robots never move to an occupied cell, such a collision must involve either Follower or Leader robots that move to the same cell simultaneously. Now, the movements of a Follower is restricted to the part of the path between its current cell and its predecessor which is always at least three cells away and no other robot can enter this path as all exits are blocked. Thus, a Follower robot is never involved in a collision. This leaves us with only Leader robots, and since there is at most one such robot at any time, there cannot be any collisions. ■

Proof of Proposition 3
First notice that any communication process ends within a finite time. Outside of the communication process, let us consider the cases when a robot waits for another robot. Note that Stopped robots have already terminated and a blocking robot never waits for another blocking robot, so any deadlock must involve at least a Leader or Follower robot. The only case when a leader waits is when its successor s is at distance four. But in this case successor s would move unless the successor of s is also at a distance four from s. This implies that every robot in the path has its successor at distance four. Otherwise the robot that is closer than distance four from its successor, would move, contradicting the existence of deadlock. However, since the robot at the door does not have any successor, it could move and this contradicts the existence of a deadlock. Thus no leader or follower could be involved in a deadlock and thus there are no deadlocks during the algorithm. ■

Proof of Proposition 4
We have already seen that the algorithm ensures there are no cycles in the path and thus no robot visits the same cell twice except during a communication process. A communication process always ends in finite time, and at that both the receiver and the sender are back in their original positions. Moreover, according to the rules of the algorithm no pair of robots communicates more than twice from the same cells. So, within a finite time the robots involved in a communication, must either moves to a new cell or stop moving. Since there are only a finite number of new cells, all robots must stop after a finite time. ■

Proof of Proposition 5
Suppose for the sake of contradiction that after the algorithm terminates there are some empty cells and let us consider an empty cell p that is adjacent to an occupied cell q (such a cell must exist as S is connected). Since the algorithm has terminated, the robot r in cell q is in state *Stopped*. This implies that when r became Stopped, either cell p was occupied or it was the

previous location of robot r. In either case cell p had been occupied by some robot in the past. Consider the last robot r_l to visit p. When robot r_l moved out of cell p, it must have a successor and thus the successor must have visited cell p too. This contradicts the fact r_l was the last robot to visit cell p. Thus, there cannot be any empty cells after the termination of algorithm **MUTE**.

■