Uniform dispersal of robots with minimum visibility range

Attila Hideg[1] and Tamás Lukovszki[2]

[1] Department of Automation and Applied Informatics,
Budapest University of Technology and Economics, Budapest, Hungary
attila.hideg@aut.bme.hu
[2] Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary
lukovszki@inf.elte.hu

Abstract. We consider the filling problem, in which autonomous mobile robots enter a connected orthogonal area from several entry points and have to disperse in order to reach full coverage. The entry points are called doors. The area is decomposed into cells. The robots are autonomous, anonymous, they have a limited visibility range of one unit, and do not use explicit communication. Collision of the robots is not allowed. First we describe an algorithm solving the filling problem for the single door case in $O(n)$ time steps in the synchronous model, where n is the number of cells in the area. This algorithm is optimal in terms of visibility range, and asymptotically optimal in running time and size of persistent memory used by the robots. Moreover, we show that our algorithm solves the multiple door filling problem in $O(n)$ time, as well. For the multiple door case, our algorithm is asymptotically worst-case optimal, and its running time is at most k times the running time of the optimal algorithm for any input, where k is the number of doors.

1 Introduction

In swarm robotics a huge number of simple, cheap, tiny robots can perform complex tasks collectively. The greatest advantages of such systems are scalability, reliability, and fault tolerance. Contrary to a single-robot system, which requires complex, expensive hardware and software components with redundancy, the same attributes can be achieved by simply adding more robots to a multi-robot system. In case of mobile robots, the spatial distribution of the robots has huge advantage in problems related to exploration, coverage, demining, toxic waste cleanup, etc. In this paper we study the *uniform dispersal* (or *filling*) of synchronous robots in an unknown, connected area.

The area is decomposed into cells and the robots are injected one by one into the area through an entry point, which is called the *door*. The robots have to reach full coverage by occupying all cells. This problem is called filling, and was introduced by Hsiang et al. [5]. When more than one door is present in the area the problem is called *multiple door filling* or *k-door filling*.

In [5] the goal was to achieve a rapid filling, minimizing the *make-span* (time to reach full coverage) of the algorithm. Their solution required $2n - 1$ cycles to reach full coverage, where n was the number of cells in the area.

Barrameda et al. [1] investigated the minimum hardware requirements and the possibilities of solving the filling problem by robots with constant visibility radius, communication range, and constant number of bits of persistent memory.

The algorithms in [1, 2, 5] used the *Leader-Follower* method, where one robot is elected as a leader and the rest of the robots follow it until the leader is blocked, then the leadership is transferred to another robot. The leader-follower method results in a DFS-like dispersion in the area. Collisions are prevented through the property that the leader explores new cells (cells that were never visited before) and the followers simply moving towards their predecessor (the robot they are following).

In case of the multiple door filling problem the robots enter through multiple doors and there are several leaders in the area. In [1] the robots were colored according to the door they entered, and the robots required to have their color visible to other robots within the visibility range. In [3] Das et al. showed that allowing visible colors or lights yields a more powerful computational model than allowing infinite visibility range but no lights.

In this paper a fundamental question is, whether it is possible to reduce the hardware requirements of the robots and still fill an unknown connected orthogonal area, even in presence of holes, and maintain $O(n)$ runtime. These hardware requirements are: visibility range, size of persistent memory, and avoidance of explicit communication and the usage of lights.

1.1 Our Contribution

We present a method for filling an unknown, connected orthogonal region S consisting of $|S| = n$ square shaped cells by a set of n autonomous anonymous robots with a visibility radius of 1 hop in $O(n)$ time in the synchronous computational model. The robots require $O(1)$ bits of persistent memory and cannot communicate (they do not use explicit communication, nor colors or lights). The only precondition is that they require a common coordinate system.

First, we consider the single door case, and present an algorithm which solves the problem without collisions and terminates in $O(n)$ time. Then, we show that the presented approach solves both the single door and multiple door filling problem in orthogonal areas.

Regarding this model our algorithm is optimal in terms of visibility range and asymptotically optimal in the size of the memory. Moreover, it is asymptotically optimal in running time in the single door case, asymptotically worst-case optimal in the multiple door case, and its running time is at most k times the running time of the optimal algorithm for any input. The optimality regarding the visibility range follows from the fact that with a visibility range less than 1 the robots cannot even distinguish between occupied and unoccupied neighboring cells. The asymptotic optimality of the memory size $O(1)$ follows from the result by Barrameda et al. [1]; they proved that oblivious (memoryless) robots cannot deterministically solve the problem. The asymptotic optimality of the running time $O(n)$ follows from the fact that we can place one robot per round in the single door case and n robots must be placed. For the asymptotically worst-case optimality for the k-door case, we show inputs where almost all robots must pass through a single cell to get into a large component of the area. This single cell behaves similarly to a single door, and the dispersion must take $\Omega(n)$ time.

Organization: In Section 2 we define our model. In Section 3 we present previous results on the filling problem and related problems. Then in Section 4 we describe and analyze our algorithm for the single door and for the multiple door cases. Finally, Section 5 summarizes the paper.

2 Model

We are given an orthogonal area, i.e. polygonal with sides either parallel or perpendicular to one another, which is decomposed into equal sized, square shaped cells (see [2]). The size of each cell allows only one robot to occupy it at any given time. Each cell has at most four adjacent cells in fixed directions: North, South, East, and West.

The robots' actions are divided into three phases: *Look*, *Compute*, and *Move*. During the Look phase, the robots take a snapshot of their surroundings, in the Compute phase they perform their computations (e.g. which action should they perform), and during the Move phase they move there. This is called the Look-Compute-Move (LCM) model, which is commonly used in distributed robotics.

When the robots perform their LCM cycles at the same time, the model is called fully synchronous (FSYNC). In the FSYNC model, each robot takes snapshots at the same time, compute, and move at the same time based on a global tick.

The robots are *anonymous*, i.e. they cannot distinguish each other, and are equipped with limited hardware. They have a visibility range of 1 hop, i.e. each robot can 'see' only the cells which they are occupying and the cells adjacent to it. In one LCM cycle a robot can move to one of its neighboring cells or stay at place. The robots are *silent*, i.e. they cannot communicate at all. They are *finite-state* robots, i.e. they have a constant number of bits of persistent memory. The robots have a common notion of North, South, East, and West.

The entry points, called *doors* are always occupied by a robot. Whenever a robot moves from a door cell, a new one is placed there. The doors cannot be distinguished from other cells by the robots, moreover, the robots do not know which door they used to enter the area.

3 Related Work

Hsiang et al. [5] investigated the make-span (i.e. the time to reach full coverage) of filling of a connected orthogonal region measured in rounds. They assumed that robots have a limited ability to communicate with nearby robots, i.e. a robot is able to exchange a constant-size message. They proposed two solutions, BFLF and DFLF, both modeling generally known algorithms: BFS and DFS. In DFLF, the method maintained a distance of 2 hops between the robot and its successor. As a consequence the method only required visibility range of 2 hops. However, the robots had to be able to detect the orientation of each other.

Barrameda et al. [1] assumed common top-down and left-right directions for the robots and showed that robots with visibility range of 1 hop and 2 bits of persistent memory can solve the problem in an orthogonal area if the area does not contain holes, without using explicit communication. Holes are cells in the area which cannot be occupied by robots (e.g. obstacles).

In [2] Barrameda et al. presented two methods for filling an unknown orthogonal area in presence of obstacles (holes). Their first method, called **TALK**, requires a visibility range of 1 hop if the robots have explicit communication. The other method, called **MUTE**, do not use explicit communication between the robots, but it requires visibility range of 6. Both methods need $O(1)$ bits of persistent memory.

For the multiple door case, in [1] a method, called **MULTIPLE**, has been presented. It solves the problem for robots with visibility range of 2 hops, no explicit communication and a $O(1)$ bits of persistent memory. In this solution the doors are colored with different colors and the robots are colored according to the door they enter. The color of the robots is visible to other robots within the visibility range.

A summary of these previous results and a comparison to our contribution is presented in Table 1.

Method	Visibility range (hops)[1]	Comm. range (hops)[1]	Memory (bits)	Area
DFLF [5]	2	2	2	Arbitrary
TALK [2]	2	2	4	Orthogonal
MUTE [2]	6	0	9	Orthogonal
MULTIPLE [1]	3	0	4	Orthogonal
our method: (here)				
Single door	1	0	13 bits	Orthogonal
Multiple doors	1	0	13 bits	Orthogonal

Table 1. Summary of the requirements of Filling algorithms.

In [6] a related problem, the pattern formation problem has been investigated in the FSYNC model. In the pattern formation problem n robots are placed arbitrarily in the 2 dimensional grid and they have to form a given connected pattern F, known for all robots. In [6] a common coordinate system for the robots

[1] In [2], grid cells sharing a common edge or a common corner with the current cell c of the robot are assumed within the visibility range of the robot. In our model a cell sharing only one corner with c has a hop distance of two. Thus, it is outside of the visibility range. Only the cells sharing a common edge have hop distance of one.

has been assumed. The presented solution requires no explicit communication, a visibility range of 2 hops, and 2 bits of persistent memory. The robots are gathered in a certain point. They reach this point one-by-one and then start to traverse a spanning tree of F and fill the tree by this traversal. The pattern formation algorithm needs $O(|F| + d)$ rounds, where d is the diameter of the initial configuration. Only considering the filling phase of this algorithm, it needs $O(|F|)$ rounds.

For an excellent overview of distributed mobile robotics, we refer to the book by Flocchini et al. [4].

4 Method

In this section we describe the algorithm for the filling problem, i.e. we define the states and state-transitions of the robots. Then we prove that the algorithm solves both the single door and multiple door filling problem without collisions.

4.1 Concept

Our algorithm is based on the leader-follower approach, which means a certain robot is the leader while the rest are following it. This is a common approach to eliminate collisions, as the leader is the only robot which is capable of moving to cells, that were never occupied before. Such cells are called *free cells*. Each follower has a *predecessor* which is the robot it is following during the dispersion, and each robot, except the one at the door, has a *successor* which follows it.

Each robot stores its state, which can be one of the following: *None*, *Leader*, *Follower*, *Stopped*. The state of the robot is not visible for other robots. The transitions between these states are shown in Fig. 1. The robots placed at the door are initialized with None state. They can switch to either Leader or Follower state. In Follower state, the robot can switch to Leader state if and only if its predecessor was the Leader and that Leader switched to Stopped state. This ensures that the number of Leaders does not increase.

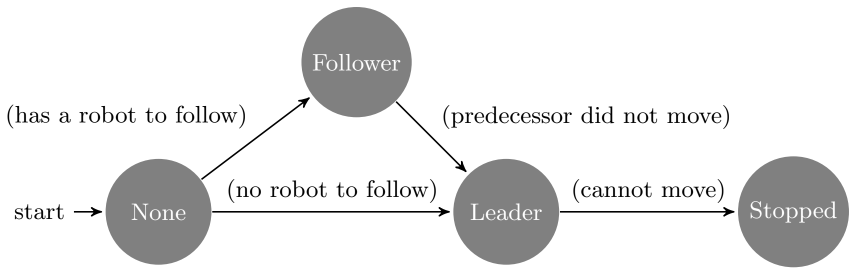

Fig. 1. States and state-transitions of the robots. The edges are labeled with the condition for the transition.

The robots repeatedly perform their LCM cycles. We label each cycle with a direction and call it a *step*. There are four possible directions: North, East, West, South; the steps are labeled correspondingly N-step, E-step, S-step, W-step. In each step the robots can only move toward the cell in the corresponding direction. A *round* is a sequence of four consecutive steps starting with an N-step, followed by an E-step, followed by an S-step, ending with a W-step (see Fig. 2). The rounds and the steps start synchronously at the robots.

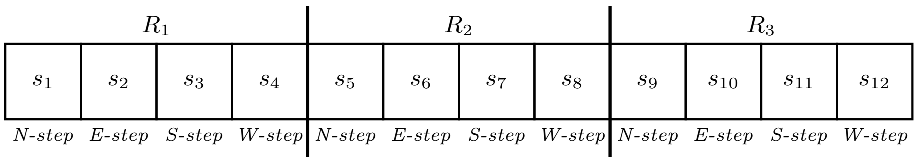

Fig. 2. The structure of the rounds. Three rounds (denoted by R_1,R_2,R_3), each consists of four consecutive steps in fixed order: step s_1 is an N-step, s_2 is an E-step and so forth.

After a robot r is placed, in the first round (and then in every odd round) it always observes its surroundings, then it can only move during the second round (and during every even round), irrespective of its current state. Note: odd and even rounds are relative to the starting round for r.

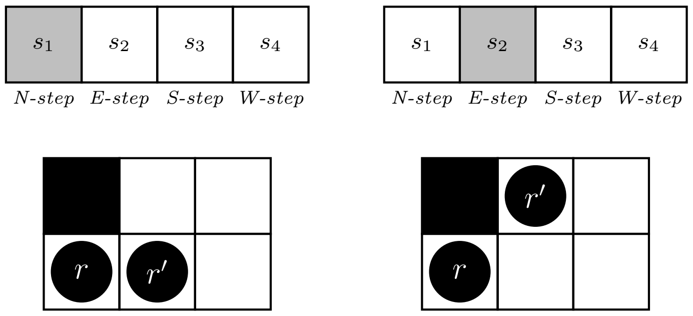

Fig. 3. Robot r following r'.

Our algorithm mimics a DFS exploration of the unknown region simultaneously started from the doors. A current Leader defines a path in the DFS tree from the root (door) it entered, which path is traversed by it. When the Leader switches to Stopped state it is in a leaf of the DFS-tree. The Follower robots will fill the path segment between the last branch vertex and the Leader (in the leaf). Then the leadership transfers backwards until we reach a branch vertex. Then the robot on that branch vertex becomes a new Leader and traverses to the next branch. As the destination selection is deterministic, the robots on branch vertices are checking the neighbors in the same order, even without the explicit knowledge of the DFS-trees, or if the predecessor becomes Stopped. Therefore, they will move in the direction of the first free branch.

The main difficulty to realize this concept is that it must be ensured during the algorithm that the Leaders are able to determine which neighboring cells are free and the Followers must know where their predecessors are, despite the fact that the predecessors can be outside of the 1 hop visibility range. Now we describe the exact behavior of the robots in the different states.

Stopped: If a robot r is in Stopped state it does not move anymore. Other robots will treat it like it would be an obstacle.

Follower: If a robot r is a Follower it follows its predecessor r' until r' stops for two consecutive rounds (r' is not able to move in its even round). Then the leadership will be transfered from r' to r. In the odd rounds of r it checks the occupancy of the cell where its predecessor is. When r takes a snapshot in the Look phase in its odd rounds it always sees its predecessor in the N-step. If r' moves in a certain step, then r will not see r' in the next step. E.g., if r' moves to north in the N-step, then after the Look phase of the E-step r does not see r', but it knows that r' moved to the north. Therefore, r knows where r' is, even if the distance between them is 2 and r does not have r' in its visibility range. In the next round r occupies the former cell of r' and r' becomes visible again. In every even round r moves toward its predecessor, occupying the previous cell of r' in the step corresponding to the direction r moves. As a result, the successor of r also knows which direction r moves to and follows r.

There might be a case during the odd round of r, that r' does not move in that round. It is only possible if r' does not have any cells it can move to, either because it is surrounded by obstacles or other robots. It implies that r' switches to Stopped state and r can switch to Leader state after this round (more precisely, after the Look phase of the N-step of the next round, when r recognizes that r' did not move in the odd round of r). We refer to this event as the *transfer of the leadership*. It guarantees that r can only switch to Leader state after its predecessor r' switched to Stopped state. Therefore, the invariant that there can only be one Leader at any time (or k Leaders in the multiply door case) is fulfilled. Note: r switches to Leader state only in the N-step of its even round. Then in that round r performs the actions of robots in Leader state.

Leader: If r is a Leader it moves to a free cell in every second (even) round. In each step of every odd round r checks each of its neighboring cells if it is occupied. Occupied cells can not be free and they can be excluded from the set of potential moving directions in the next (even) round. In the next (even) round in each step r checks the neighboring cell c corresponding to the direction of the step again. If c is unoccupied and it has been unoccupied in the previous (odd) round, r moves to c and r does not check the remaining directions in that round. If r cannot move in any direction it switches to Stopped state and terminates its actions. Note: r can switch to Stopped state in the same round it switched to Leader state.

None: After r has been placed at a door in round R_i, its initial state is None. It only has to know, which step is currently performed by the system, i.e. an N-, E-, S-, or W-step. If the current step is an N-step and no robots are visible, then r is the first robot placed at that door and it has to switch to Leader state. In every other scenario, its predecessor robot r' is in the direction corresponding to the previous step, e.g., if the current step is an E-step then the predecessor r' leaved in the previous step, which is an N-step. Therefore, it must be in the northern neighbor cell. Let $R_{i'}$ be the round in which r' has moved from the door. Note that $i' = i - 1$ if r is placed in an N-step, and $i' = i$ otherwise. If $i' = i$ then round R_i is a shortened round for the newly placed robot r and r does not perform any actions in this round. Then the first round of r starts with the next N-step.

As the predecessor r' of the newly placed robot r has moved during round $R_{i'}$, round $R_{i'+2}$ is the next even round for r' (i.e. the next round r' moves). We have to ensure that $R_{i'+2}$ will be an odd round round for r. Therefore, the only action r performs in $R_{i'+1}$ is switching in Follower state at the end of the round. The first round when r performs its actions is $R_{i'+2}$, which is an odd round for r, and it acts as a Follower. Note: in $R_{i'+3}$, robot r can switch to Leader state if it recognizes that r' is in Stopped state, and r can switch even to Stopped if r cannot find any free cells.

4.2 Single Door Case

First, we show that the described algorithm fills the area without collisions in the case, when the robots are entering in a single door.

Lemma 1. *No collisions can occur during the dispersion.*

Proof. In each step each cell can only be occupied from one sole direction. Even if two robots would move to the same cell during the same round, only one will move there, depending on the direction it wants to enter. □

Lemma 2. *During each step each Follower knows where its predecessor is.*

Proof. Consider a Follower r. Let r' be its predecessor. Based on the timing of the movement, i.e. in which step r' moved, r knows the target cell of r'. With other words, when a robot r sees its predecessor r' on cell c during step s_i, but if r detects c is unoccupied in s_{i+1}, it implies that r' has moved during s_i. The robot r' can only move to one direction which corresponds to the direction of s_i. Consequently, after the Look phase of s_{i+1} the robot r knows where its predecessor r' is, even if r' is outside of its visibility range. In the next round r moves to c and r' becomes visible again.

The argument above does not apply when r' moves from the door, as r has not been placed yet. However, r knows, in which step it has been placed. Let s_i be the step in which r' has moved from the door and s_{i+1} the step, in which r has been placed. Then r' had to move in the direction corresponding to step s_i. Therefore, r knows where r' is. □

Lemma 3. *The Leader only moves to free cells.*

Proof. The leader has to move to free cells to increase the coverage with each of its movement and to prevent infinite loops. Based on one single snapshot the Leader r can only distinguish between occupied and unoccupied cells. Let R_i be the round in which r has moved (its even round), then R_{i+1} will be its odd round. In R_{i+1}, if a neighboring cell is occupied at any time during R_{i+1}, r recognizes it as not being free. In the next (even) round R_{i+2} in each step r checks the neighboring cell c corresponding to the direction of the step again. If c is unoccupied and it has been unoccupied in the previous round, r recognizes c as a free cell and moves to c. Then r does not check the remaining directions in that round.

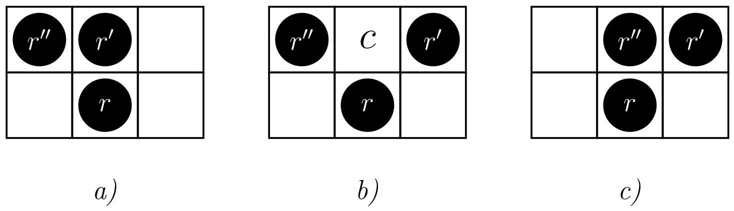

Fig. 4. A non-free cell c and its neighborhood in three consecutive rounds.

Assume r would like to determine whether a neighboring cell c is free in R_i, and assume c is not a free cell. There are two cases: c contains a robot at the beginning of R_i (Fig. 4.*a*) and *c*)) or not. If c is occupied, then r recognizes it as not being free and will not try to move there. If c is unoccupied at the beginning of R_i and it is not a free cell (Fig. 4.*b*)), there was a robot r' in c at the beginning of R_{i-1} and moved from it to a neighbor of c which is not visible by r. Then, r'' (the successor of r') will enter c in R_i to follow r'. At the beginning of the next round (R_{i+1}), c will be occupied by r'' (as in Fig. 4 *c*)) and r will not move there.

As a result two consecutive rounds, R_i and R_{i+1}, are sufficient for r to identify c as a non-free cell. In other case, c is identified as a free cell and r can move to c in R_{i+1}. □

Lemma 4. *At most one Leader is present in the area.*

Proof. Recall the event *transfer of the leadership.* The Leader r switches to Stopped state during R_i. Its successor r' can only switch to Leader state once it has detected that r is in Stopped state, which will be in the first step of R_{i+1}. □

Lemma 5. *The proposed method fills the area.*

Proof. For contradiction, assume a cell c is left unoccupied after the algorithm is terminated. There are two cases: *i*) c is unoccupied but not free (some robots visited c already but c left it unoccupied), *ii*) c is a free cell.

For i), assume r is the last robot which left the cell c. The successor of r must have followed r and occupied c, which is a contradiction.

For ii), consider a cell c' which is a closest occupied cell to c. There is a path between c and c' consisting only of unoccupied cells (or c' is a neighbor of c). By i) all unoccupied cells are free cells, as well. Let r be the robot in c', which is in Stopped state as the algorithm has been already terminated. However, r has a free neighboring cell, therefore, r can move there. This prevents r from switching to Stopped state, which is a contradiction.

□

Theorem 1. *By using the presented algorithm a connected orthogonal area with a single door is filled in $O(n)$ rounds without collisions by robots with visibility range of* 1 *hop and $O(1)$ bits of persistent memory.*

Proof. According to the previous lemmas, the algorithm fills the area and prevents collisions. The robots move in their even rounds. Whenever a robot r is placed at the door, its first odd round is started with the first N-step, i.e. if r is placed in a E-, S-, or W-step, it waits for the next N-step. Then in the odd round it just observes and in the next even round it moves, allowing the placement of a new robot. Therefore, a new robot can be placed in every third round. Since the number of cells in the area is n, placing n robots requires $3n$ rounds, which takes $12n$ LCM cycles.

The persistent memory of the robots must store the state of the robots (2 bits), the parity of the current round (1 bit), and the step of the current round (2 bits). Leaders must additionally store information about the occupancy of the neighboring cells observed in the odd round (4 bits). Followers must store the direction they will move in their next even round (2 bits), which is the direction their predecessor is in their odd round. Additionally, Followers must store where the predecessor moves in their odd round (2 bits) and the information about the occupancy of the neighboring cells for the case when the Follower has to switch to Leader state. □

4.3 Multiple Door Case

In the multiple door case, also called k-door filling, the robots are entering the area through $k \geq 1$ doors. The $k = 1$ case is the single door case. In the k-door filling we assume that each door has enough robot, thus, when a robot leaves a door, a new one can be placed there.

During the analysis of the multiple door case, we examine how multiple Leaders and the set of Followers following them interact with each other. We prove that robots entering from distinct doors cannot collide or block each other.

An invariant of the algorithm is that each Follower only follows its predecessor or becomes a Leader. First, we have to prove that robots in Leader state are not colliding with other robots. Then, we have to show that the paths of the Leaders do not cross each other (for the single door case, we have already shown that the leader does not cross its own path).

Lemma 6. *A Leader cannot collide with a Follower.*

Proof. Lemma 3 still holds: each robot in Leader state can determine which cells are free within two rounds. Therefore, they will not go to cells where a Follower would. □

Lemma 7. *A Leader cannot collide with another Leader.*

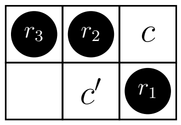

Fig. 5. Leaders can not collide. The Leader r_1 occupies c in the N-step. In the E-step r_2 sees c as an occupied cell. An important aspect is that in this scenario the robot r_2 still has to move to a cell c' different from c, if it can. Then it remains a Leader. If c' is not free, then r_2 cannot move and in step s_4 and switches to Stopped state.

Proof. Assume a scenario where two leaders r_1 and r_2 would move to the same cell c (see Fig. 5). Furthermore, assume both r_1 and r_2 are in their even round and recognized c as free cell. Let R_i be this round, consisting of steps $s_1 \dots s_4$, and as R_{i+1} is their even round, they both will try to move there. Since c is visible by r_1 and r_2 from different directions, therefore they try to move to c in different steps, which means one of the robots (w.l.o.g. r_1) will move and the other one (r_2) will be prohibited to move to c after it has been occupied by r_1. □

Lemma 8. *Paths of different Leaders cannot cross each other.*

Proof. Lemma 3 still holds for the multiple door case. It means that the Leader only moves to free cells, which implies that a Leader will not cross the path, where Followers are already moving. In Lemma 7 we have shown that two Leaders cannot collide (and will not cross the path of each other). Therefore, the path of the Leaders will not cross each other during the dispersion.

Theorem 2. *By using the presented algorithm a connected orthogonal area with multiple doors is filled in $O(n)$ rounds without collisions by robots with visibility range of* 1 *hop and $O(1)$ bits of persistent memory.*

Proof. As in the single door case, the robots do not collide with each other. Furthermore, the paths of the Leaders cannot cross, they do not block robots entering from other doors. □

Lemma 9. *The algorithm is asymptotically worst-case optimal in the multiple door case.*

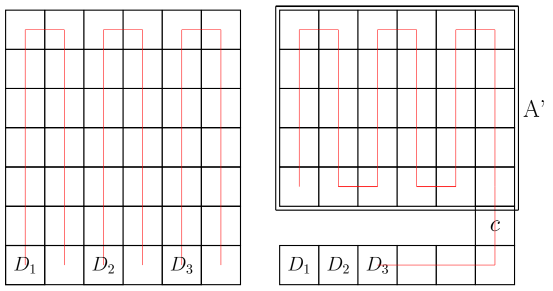

Fig. 6. Best-case (*left*) and worst-case (*right*) examples for the make-span of multiple door filling. The best-case is, when the robots entering at disjoint doors fill equal sized partitions of the area. A worst case input, when single door is used to fill almost the whole area. D_1, D_2, D_3 are the doors. Red represents the path of the first Leader entering through the given door.

Proof. Consider a region A, as depicted in Fig. 6 b). Removing the cell c from A disconnects the region, such that all doors are separated from the major connected component A' of $A \setminus c$. Therefore, all robots must enter to A' through c. Since collision is not allowed, only one robot per round can enter to A' in any algorithm. If A' contains $\Omega(n)$ cells, then each algorithm requires $\Omega(n)$ rounds. Our algorithm fills A in $O(n)$ time. □

For any input, even in the best-case, at most k robots per round can be placed in the area. Therefore, $\Omega(n/k)$ is a lower bound on the filling time of any algorithm for each input. Consequently, for each input the $O(n)$ running time of our algorithm is at most k times the running time of the optimal algorithm.

Note: for certain inputs our algorithm also reaches k speedup (see Fig. 6.a)), When robots entering from different doors fill equal sized partitions of the region, the runtime is $O(n/k)$.

5 Summary

We have considered the filling problem in an unknown, connected, orthogonal region, where the robots enter the region through several doors and they have to disperse in order to reach full coverage. The robots are autonomous, anonymous, silent, they have a limited visibility range of 1 hop, and use $O(1)$ bits of persistent memory. They have a common notion of North, South, East, and West. Collision of the robots is not allowed. We have presented an algorithm solving the filling problem with multiply doors in $O(n)$ time steps in the synchronous model, where n is the number of cells in the area. This algorithm is optimal in terms of visibility range, and asymptotically optimal in size of persistent memory used by the robots. The running time is asymptotically optimal for the single door case. For the multiple door case, our algorithm is asymptotically worst-case optimal, and its running time is at most k times the running time of the optimal algorithm for any input, where k is he number of doors.

References

1. Barrameda, E.M., Das, S., Santoro, N.: Deployment of asynchronous robotic sensors in unknown orthogonal environments. In: Algorithmic Aspects of Wireless Sensor Networks: 4th Int. Workshop (ALGOSENSORS 2008), Revised Selected Papers. pp. 125–140 (2008)
2. Barrameda, E.M., Das, S., Santoro, N.: Uniform dispersal of asynchronous finite-state mobile robots in presence of holes. In: Algorithms for Sensor Systems: 9th Int. Sym. on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2013), Revised Selected Papers. pp. 228–243 (2014)
3. Das, S., Flocchini, P., Prencipe, G., Santoro, N., Yamashita, M.: Autonomous mobile robots with lights. Theor. Comput. Sci. 609, 171–184 (2016)
4. Flocchini, P., Prencipe, G., Santoro, N.: Distributed Computing by Oblivious Mobile Robots. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers (2012)
5. Hsiang, T., Arkin, E.M., Bender, M.A., Fekete, S.P., Mitchell, J.S.B.: Algorithms for rapidly dispersing robot swarms in unknown environments. Algorithmic Foundations of Robotics V, Springer Tracts in Advanced Robotics 7, 77–93 (2004)
6. Lukovszki, T., Meyer auf der Heide, F.: Fast collisionless pattern formation by anonymous, position-aware robots. In: Proc. 18th International Conference on Principles of Distributed Systems (OPODIS). pp. 248–262 (2014)