Auton Agent Multi-Agent Syst (2009) 19:210–243
DOI 10.1007/s10458-008-9075-4

Rapid exploration of unknown areas through dynamic deployment of mobile and stationary sensor nodes

Ettore Ferranti · Niki Trigoni · Mark Levene

Published online: 31 December 2008

Abstract When an emergency occurs within a building, it may be initially safer to send autonomous mobile nodes, instead of human responders, to explore the area and identify hazards and victims. Exploring all the area in the minimum amount of time and reporting back interesting findings to the human personnel outside the building is an essential part of rescue operations. Our assumptions are that the area map is unknown, there is no existing network infrastructure, long-range wireless communication is unreliable and nodes are not location-aware. We take into account these limitations, and propose an architecture consisting of both mobile nodes (robots, called agents) and stationary nodes (inexpensive smart devices, called tags). As agents enter the emergency area, they sprinkle tags within the space to label the environment with states. By reading and updating the state of the local tags, agents are able to coordinate indirectly with each other, without relying on direct agent-to-agent communication. In addition, tags wirelessly exchange local information with nearby tags to further assist agents in their exploration task. Our simulation results show that the proposed algorithm, which exploits both tag-to-tag and agent-to-tag communication, outperforms previous algorithms that rely only on agent-to-tag communication.

Keywords Autonomous agents · Area exploration · Sensor networks · Collaboration · Tags

1 Introduction

When an emergency occurs within a building, it is crucial for the first responders to acquire as much information as possible on the ongoing situation, in order to identify and contain hazards and coordinate the rescue of victims. Initially, the area is off-limits and hazardous

E. Ferranti (✉) · N. Trigoni
University of Oxford, Oxford, UK
e-mail: Ettore.Ferranti@comlab.ox.ac.uk

M. Levene
Birkbeck College, University of London, London, UK

for anyone not wearing respiratory equipment, garments or barrier materials to protect themselves from exposure to biological, chemical, and radioactive hazards. This kind of suit can be very heavy and bulky, consequently limiting the first responders' movements, and reducing their sensing capacity (touch, vision, and hearing). A group of autonomous mobile nodes, referred to as *agents*, should therefore be deployed in the area to acquire all the information that could assist the tasks of the first responders.

Exploring all the area in the minimum amount of time and reporting back to the human personnel outside the building is an essential part of rescue operations. However, such operations may be obstructed by a number of limitations, e.g. the possible lack of a terrain map (the environment could in any case be substantially changed as the result of a disaster), the failure of previously established networks, and the short-range and often unreliable wireless indoor communication. In addition, it might be difficult to use GPS positioning inside a building, so an agent cannot rely on knowledge of its exact location within the terrain, even if it were able to memorise its previous steps.

In this paper, we take into account these limitations, and assume that agents can rely only on local information that is sensed in their vicinity (which other agents have left behind them as a trace), before making the next exploration step. We propose an approach to area exploration in which a swarm of *agents* enter the emergency area and dynamically deploy a network of stationary sensor nodes, referred to as *tags*, in order to label the environment. Agents do not communicate directly with each other; instead, they coordinate indirectly by leaving traces of information on the tags that they deploy on the space. Agents are able to read and update the state of local tags, and by doing so, they leave valuable information for other agents in order to help them make intelligent navigation decisions. In addition, the set of deployed tags form a multi-hop wireless network. Tags disseminate their local information to other tags downstream hop-by-hop in order to further assist the agents in their exploration task.

The feasibility of the approach in real-time systems has been confirmed in Batalin and Sukhatme [1–4], who used radio beacons to guide the navigation of robots and assist them in the coverage of an unknown terrain, and in O'Hara et al. [5–8], who deployed an extensive test-bed of small sensors (GNATs) to guide the navigation of a LEGO Mindstorm/RCX robot using infrared transmitters and receivers. The goal of the present work is therefore to provide a more sophisticated and better performing exploration algorithm to be used by the agents.

Furthermore, a very accurate formalisation of the online exploration problem has been provided by Wagner et al. [9] and it would thus be redundant within the scope of the current work.

The key contribution of our proposed exploration algorithm, named HybridExploration, is that it combines two modes of communication: between agents and tags (agent-to-tag) and multi-hop communication within the stationary network of tags (tag-to-tag). It is fully distributed and does not require centralized control of the agents to determine their next move. It only exploits short-range communication between agents and tags, and among tags, and does not use unreliable long-range communication among agents, or agents and the human responders.

In this paper, we compare HybridExploration with three competing algorithms, namely Ants [10], Multiple Depth First Search (MDFS) [11–14], and Brick & Mortar [14]. In Sect. 5.4, we also briefly describe another closely related algorithm, CLEAN [9], which is very similar to Brick & Mortar. These algorithms make similar assumptions to those discussed above, i.e. no knowledge of the area map, lack of GPS positioning, and no centralized control of agents. Agents dynamically deploy tags on the floor and, by reading and updating the state of the local tags, they coordinate their exploration task. However, unlike

HybridExploration, these algorithms do not exploit tag-to-tag communication, i.e. they do not use the multi-hop communication capabilities of the deployed sensor network.

Other weaknesses of these algorithms, compared to HybridExploration one, are as follows:

1. Agents running the Ants algorithm cannot determine when the exploration task is completed. Moreover, whilst the first few agents rapidly discover new terrains, most of the remaining ones may dwell on already explored network areas, leading to inefficient use of agent resources.
2. Agents running MDFS know when the exploration task terminates, but their poor coordination leads toward long exploration times.
3. Agents running Brick & Mortar use a complex loop resolution mechanism that significantly delays the task of area exploration, especially in topologies with many obstacles (e.g. desks in the middle of an open space).

Our proposed algorithm, named HybridExploration, overcomes the limitations of existing approaches, and offers significant performance gains in terms of exploration time for a variety of terrain topologies. Our experimental results allow us to understand the impact of several parameters on the performance of HybridExploration (and competing algorithms), including the number of agents, the terrain size, the numbers of rooms, and the number of obstacles (e.g. desks or hazards in the middle of rooms).

The rest of the paper is organized as follows. Section 2 presents our model, together with its objectives and assumptions, and Sect. 3 describes the new HybridExploration algorithm. Section 4 presents a thorough experimental analysis of the proposed algorithm and the three competing approaches. An overview of related work is provided in Sect. 5, followed by conclusions and directions for future work in Sect. 6.

2 Model

We consider the task of exploring a hazardous terrain using a group of autonomous *agents*. We assume a very simple model of the area, in which the environment is divided into a grid of square cells, whose size depends on both sensing coverage and communication range of the agent. In particular, when an agent is at the centre of a cell, it must be able to cover the entire area with a sensor attached to it to scan for victims or hazards. Therefore, the size x of a cell must be determined by the range of the sensor (r_{sense}) and by the communication range of the agent (r_{comm}), which is also equal to the range of a laser sensor to detect obstacles. In particular, if we refer to Fig. 1, it must be that $x \leq \frac{2r_{sense}}{\sqrt{2}}$. Furthermore, since the agent must be able to communicate with wireless nodes, referred to as *tags*, in everyone of the 8 cells around it, and to scan the same cells for obstacles, it must be that $x \leq \frac{2r_{comm}}{3\sqrt{2}}$.

A cell can be in one of the following states:

– *Wall*: The cell cannot be traversed by an agent because it is blocked by an obstacle. In particular, we assume that an agent is equipped with sensors (e.g. laser, sonar, etc.) that enable to detect if it can successfully traverse the cell area from the center of it toward points A, B, C, D (Fig. 2) which represent all the possible accesses to the adjacent cells. If any one of these routes is blocked by an obstacle, then the cell is considered to be a wall.
– *Unexplored*: No agent has been in the cell yet, and therefore no tag has been deployed there yet.

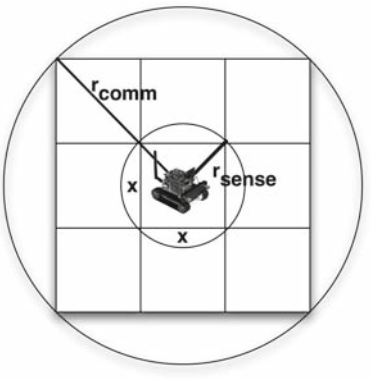

Fig. 1 Example of an agent equipped with a sensor. The sensor must cover the entire area of the cell, and the agent must be able to communicate with the 8 cells around it

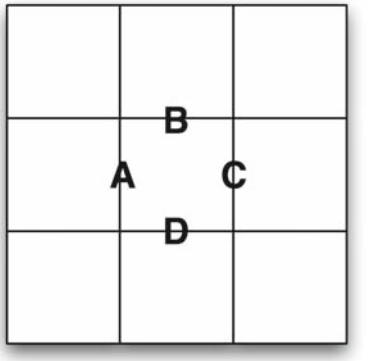

Fig. 2 A, B, C, D are the access points from the current cell toward the adjacent ones

– *Explored*: The cell has been traversed at least once, but the agents might need to go through it again in order to reach other *unexplored* cells. A tag is already deployed there by the agent who first visited the cell.
– *Visited*: The agents have already explored the cell, and they do not need to go through it again to reach other cells. Conceptually, this state is equivalent to a *wall* cell, in that no agent is allowed to traverse it. A tag is already deployed in the cell.

Furthermore, we assume that the perimeter of the area is always formed by *wall* cells.

In the remainder of this section, we provide a high level overview on the agent movement and tag deployment, and discuss the agent-to-tag and tag-to-tag modes of communication.

Agent movement and deployment of tags: Agents are initially deployed in one of the boundary cells and, in each step, they are able to move from the current cell to one of the four adjacent cells in the North, East, South or West directions. As they move to an unexplored cell, they deploy a miniature device (e.g. mote or RFID), referred to as *tag*, capable of storing small amounts of information about the state of the local cell. They also update the relative location of this tag, with respect to previously installed tags in adjacent cells. In indoor environments where GPS cannot be used, agents do not rely on knowledge of their exact location; once they find themselves within a cell, they can move towards one of the four directions until they reach the next cell.

Agent-to-tag communication: In emergency situations, long-range wireless communication may be intermittent and unreliable, so we assume that agents are able to communicate only by reading and updating the tags installed in the local and 8 neighbouring cells. We

thus only consider distributed exploration algorithms, in which agents make independent decisions about how to navigate through the terrain based on local state.

Tag-to-tag communication: In the proposed algorithm, HybridExploration, we introduce *virtual agents*, i.e. active messages that cause the execution of a small piece of code on the tag that receives them. Virtual agents alter the state of the local tag and are further disseminated to neighboring tags within communication range. The underlying assumption of our model is that a tag in a cell is able to communicate wirelessly with the tags in the four adjacent cells.

2.1 Objectives

We are now going to introduce two objectives, which we will use to assess the performance of the proposed HybridExploration algorithm, presented in Sect. 3.

1. *Exploration Objective*: all non-*wall* cells in the area are traversed by an agent at least once. This means that no cell is left in the *unexplored* state. When this objective is achieved, cells can be in any of the *explored*, *visited* or *wall* states.
2. *Termination Objective*: all cells in the area are either *walls* or *visited*. No cell is left in the *unexplored* or *explored* state.

By definition, the *Exploration Objective* is always achieved earlier (or at the same time as) than the *Termination Objective*. Both objectives should be achieved in the minimum amount of time, because in an emergency scenario as the one we are considering, speed is essential. The faster the *Exploration Objective* is achieved, the faster victims and hazards are identified. The quicker the *Termination Objective* is achieved, the earlier human responders can enter the area with the certainty that there are no hidden hazards. The efficiency of an algorithm can be measured by how fast it is able to achieve both the *Exploration* and *Termination Objectives*. The goal of the present work is therefore to devise an efficient algorithm that achieves both objectives in a rapid manner.

2.2 Assumptions

In this section, we clarify the assumptions of our model. Most of the currently listed assumptions will be further relaxed in future work.

1. *Perfect tag-to-tag and agent-to-tag communications*
   We assume that agents can communicate without faults with tags lying on the same cell or any of the 8 neighbouring cells. Moreover, agents deploy at most one tag per cell; the erroneous deployment of multiple tags on the same cell is not allowed.
2. *Perfect agent movement and localisation wrt tags*
   We assume that the agent is capable of moving from the centre of a cell to the centre of another cell without odometry errors. Moreover, it is able to identify which tag lies at the centre of the current cell, if any, and those that lie at the 8 cells around it.
3. *Static environment*
   We assume that the environment cannot dynamically change its topology during the exploration process.
4. *Abundant network lifetime*
   We assume that the network lifetime of the deployed tag network will not expire until agents will have terminated the exploration process. This is reasonable because the exploration process is not a long-term and repetitive application but a fast one and thus network lifetime is not our primary concern.

5. *Atomic operations*
   We assume that atomic operations are performed on tags. In particular, this means that a tag is able to perform one operation at each time, thus if several agents are willing to make the tag perform a set of operations then these are queued while waiting for the tag to finish the execution of the current operation.

3 The HybridExploration algorithm

The HybridExploration algorithm gracefully combines two parallel protocols, one followed by physical agents (robots, or simply agents) and one followed by virtual agents. A physical agent takes significantly longer to move from one cell to another (physical robot motion) than a virtual agent (message propagation). Hence, the time of completing the exploration task is measured as the minimum number of physical steps required by physical agents to explore the entire area.

In Sect. 3.1, we describe in detail the protocol that runs on the physical agents. This protocol enables them to cover the entire area of interest eventually, but it has two weaknesses. First, physical agents are often inefficient in exploring the area, as they cover the same cells multiple times, instead of focusing their efforts on *unexplored* parts of the space. Second, although physical agents eventually manage to visit every cell at least once, they are not aware when this happens, i.e. they have no indication of when the exploration task terminates. These two problems are discussed in detail in Sect. 3.2 and they motivate the introduction of the *Virtual Agent Protocol* in Sect. 3.3. The two protocols, the Physical Agent Protocol and the Virtual Agent Protocol, work in synchrony and together constitute our proposed HybridExploration algorithm.

3.1 Physical Agent Protocol

The main idea behind the algorithm followed by the physical agents is that of thickening the existing walls by progressively marking the cells that surround them as *visited* (see Fig. 3). Note that *visited* cells are equivalent to *wall* cells in that they can no longer be accessed. In the description of the algorithm, we refer to *wall* and *visited* cells as inaccessible cells, and to *unexplored* or *explored* cells as accessible cells. The algorithm aims to progressively thicken the blocks of inaccessible cells, whilst always keeping accessible cells connected. The latter can be achieved by maintaining corridors of *explored* cells that connect all *unexplored* parts of the network as shown in Fig. 3.

The Physical Agent Protocol (see Algorithm 1) consists of two discrete steps. In the *marking step*, the agent marks the current cell choosing between the *explored* and *visited* states. In the *navigation step*, the agent decides which cell to go to next.

Marking step: Every time an agent is in an *unexplored* cell (with no tags on it), it deploys a tag and updates the state of the cell, choosing between the *explored* and *visited* states. The current cell is marked as *visited* if it does not block the path between two accessible cells around it. Otherwise it is marked as *explored*. Figure 4 provides two examples of the marking step: one where the current cell is marked as *explored* (map a), and one where it is marked as *visited* (map b). In the first example, the only path between the two *unexplored* cells A and B traverses the cell in which the agent (black spot) is at the moment, thus the cell cannot be marked as *visited*. In the second example, there is an alternative path on the right hand side of the map, thus the current cell can be marked as *visited* without closing the way between A

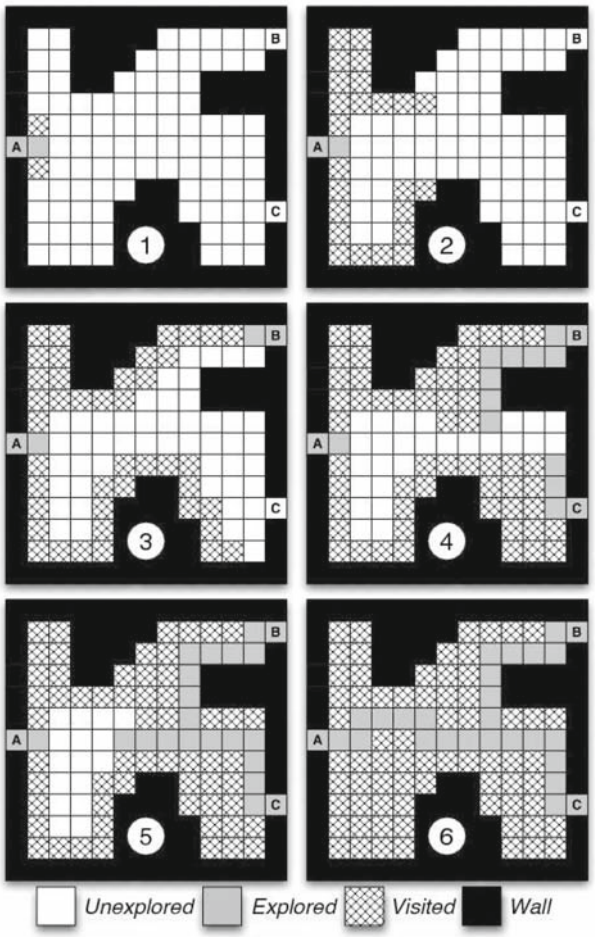

Fig. 3 Two physical agents running the Physical Agent Protocol are used to explore a room, by gradually deploying tags in its cells and updating their states. Cells A, B and C represent doors. The agents enter the room from door A and gradually thicken the walls by marking cells adjacent to walls as *visited* (Stages 1, 2 and 3). Recall that *visited* cells cannot be accessed in the future by other agents—thus, they can be viewed as virtual walls. Agents stop thickening walls if they are at risk of disconnecting two *unexplored* parts of the network. For example, in Stage 4, the physical agents create two corridors of *explored* cells in order not to disconnect *unexplored* cells in the inner part of the room from *unexplored* cells in other rooms (beyond doors B and C). When the exploration of the current room is finished (Stage 6), the cells A, B and C are connected by corridors of *explored* cells. These corridors will allow agents to traverse the room through doors A, B and C to access other *unexplored* rooms in the building. These corridors will be eventually marked as *visited* when no room in the building is left *unexplored*

and B. Note that such alternative paths are easy to compute locally, because they are strictly confined to the 8-cell perimeter of the current cell.

Navigation step: In this step, the agents take a decision about which cell to access next. Priority is always given to the *unexplored* cells which are adjacent to the current one (Fig. 5a). If the *unexplored* cells are more than one, they can be chosen at random (Fig. 5c). If the agent is equipped with a laser sensor or a sonar, and thus capable of detecting if a neighbouring

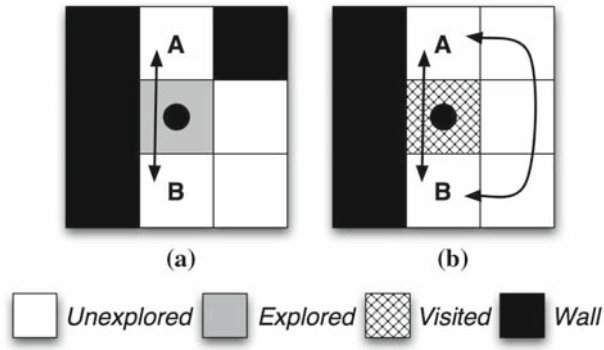

Fig. 4 Marking rules: the agent must decide to mark the current cell as *explored* or *visited*. In the first example **a** the current cell cannot be marked as *visited* because it is the only available passage between adjacent cells *A* and *B*. In the second example **b** the current cell is marked as *visited* because there is an alternative passage between *A* and *B* on the right

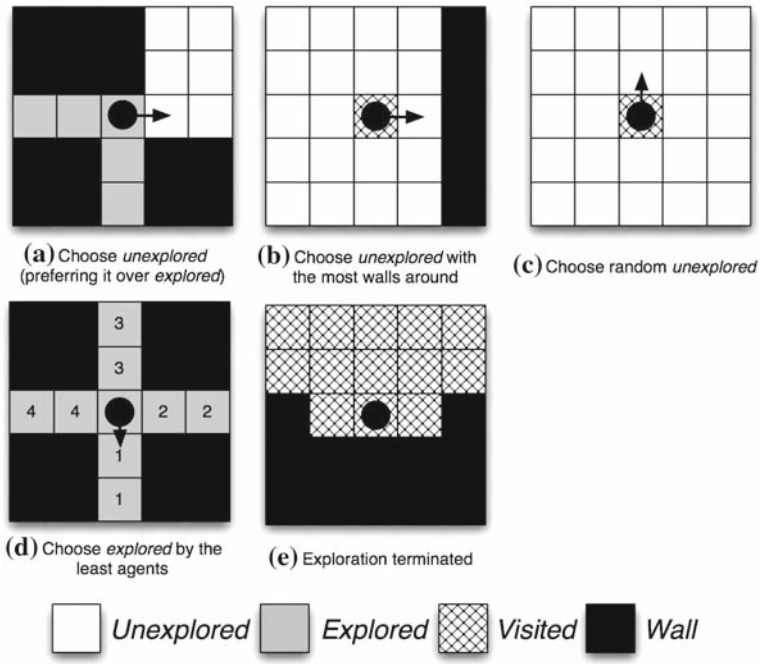

Fig. 5 Different situations in which the agent applies the navigation rules to decide which one of the adjacent cells it will go to during the next move

cell is surrounded by other black (obstacles) cells, the cell which is most likely to be marked as *visited* is chosen, i.e. the one with the most black cells around it (Fig. 5b). If there are no *unexplored* cells, the *explored* cell which has been visited the least amount of times in the past is selected (Fig. 5d). To do that, the agents simply need to increase a counter on the tag of a cell each time they traverse it, and then read all the counters of the adjacent cells and choose the minimum. In this way, an agent which traverses the same cell twice can take

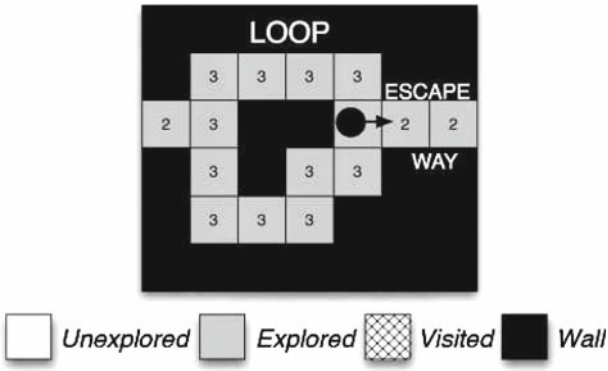

Fig. 6 The numbers in the cells represent the times the cell has been traversed in the past. The agent goes to the cell which has been traversed the minimum amount of times to avoid being trapped in a loop

different decisions about the next move instead of always going toward the same direction, thus avoiding being trapped in a loop like the one shown in Fig. 6. Finally, when the agent is surrounded by inaccessible (*wall* or *visited*) cells, it stops its exploration task (Fig. 5e).

/*Marking step */
increase the counter of the cell;
if *the cell is not blocking the path between any two surrounding cells (i.e. there exists an alternative path that connects these two cells via other surrounding cells)* **then**
  mark it as VISITED;
end
else if *the cell is UNEXPLORED* **then**
  mark it as EXPLORED;
end
/*Navigation step */
if *one of the adjacent cells is UNEXPLORED* **then**
  go to the one with the most walls around it;
end
else if *at least one of the adjacent cells is EXPLORED (and different from the cell you are coming from)* **then**
  go to the one with the minimum counter;
end
else
  stay in the current cell;
end

Algorithm 1 Physical Agent Protocol of HybridExploration

3.2 Analysis of the Physical Agent Protocol

In this section, we highlight the strengths and weaknesses of the Physical Agent Protocol. The weaknesses motivate the need to extend it with a new protocol, the Virtual Agent Protocol, which we describe in detail in Sect. 3.3. We assess the behaviour of the Physical Agent Protocol with regards to the *Exploration* and *Termination Objectives* specified in Sect. 2.1.

Advantage 1 *The Physical Agent Protocol always achieves the* Exploration Objective.

Proof Let's define a cell as *accessible* if the cell is *explored* or *unexplored*. Because of the marking rules illustrated in Fig. 4, *accessible* cells always remain part of the same group, i.e.

for each pair of *accessible* cells A and B an agent is always able to go from A to B (and vice versa) only traversing *accessible* cells. Agents always move towards an adjacent *accessible* cell with the smallest counter value (we can consider *unexplored* cells as *accessible* cells with a counter $= 0$) and, once there, they increase the local counter value. According to the proof in [15], this guarantees that agents will eventually traverse all *unexplored* cells (with counter 0) and mark them as *explored* or *visited*. □

However, the performance of the Physical Agent Protocol is severely compromised in areas with obstacles, like desks and walls in the middle of rooms. In fact, in the presence of obstacles the Physical Agent Protocol is never able to achieve the *Termination Objective*.

Let us first define the term *obstacle* formally, and then discuss the weaknesses of the Physical Agent Protocol in the presence of obstacles.

First of all, we need to specify that two cells are *linked* if one is in the 8-cell perimeter of the other. Thus, a cell can have up to 8 linked cells (in the North, East, South, West, North-East, North-West, South-East, South-West directions), whereas only up to 4 adjacent cells (in the North, East, South or West directions). Moreover, every map is always considered to be surrounded by a perimeter of *wall* cells. We can now define an obstacle O as a set of cells with the following properties:

- Each cell $c \in O$ is inaccessible, i.e. *wall* or *visited*.
- Any cell c' linked to a cell $c \in O$ belongs to the same obstacle ($c' \in O$).
- Each pair of cells in obstacle O is connected via cells of the same obstacle. That is, for any pair of cells c_1 and c_n in obstacle O, there is a sequence of cells $c_1, \ldots, c_n \in O$, such that c_{i+1} is linked to c_i, for all $i = 1, \ldots, n-1$.
- None of the obstacle cells is linked to the *wall* cells in the perimeter of the map.

Intuitively, we can think of an obstacle as an *island* of inaccessible cells separated from the perimeter of the map by other accessible cells.

The Physical Agent Protocol terminates successfully (achieving both the *Exploration* and *Termination Objectives*) only if agents do not encounter loops during the exploration process. Informally, a loop occurs when an agent traverses the same sequence of *explored* cells multiple times without being able to mark any of the cells as *visited*. Loops are encountered when there are obstacles in the middle of an area. For example, in Fig. 7a, an agent on cell C_1 of the figure will start building a corridor of *explored* cells traversing cell C_2 and then finding itself back at cell C_1 again. According to the rule in the marking step of Algorithm 1, every cell blocks the path between the previous and the following one, and is thus repeatedly marked as *explored* (Fig. 7b).

Disadvantage 1 *If there are obstacles in the area, the Physical Agent Protocol can never achieve the* Termination Objective.

Proof Initially, there are many cyclic paths of accessible (*unexplored* or *explored*) cells around each obstacle (see cyclic paths of white cells that wrap around the obstacle on the left map of Fig. 8). As agents mark more cells as *visited* the number of such paths gradually decreases (circular paths of white cells that wrap around the obstacle on the right map of Fig. 8). Assume that the Physical Agent Protocol was about to remove the last set of remaining cyclic paths, by marking a single cell as *visited*. For this to happen, this cell should be common to all remaining cyclic paths, like cell b on the right map of Fig. 8. Since this cell belongs to cyclic paths of accessible cells, by removing it, we would disconnect the accessible cells that are adjacent to it on the cyclic paths. This however is not possible according to the marking rules of the Physical Agent Protocol illustrated in Fig. 4. Hence, it is impossible to remove

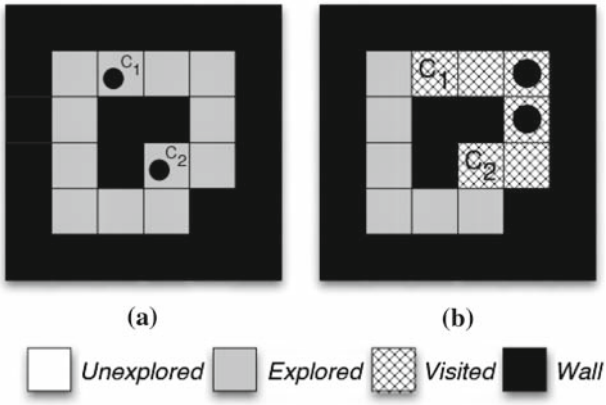

Fig. 7 The loop problem

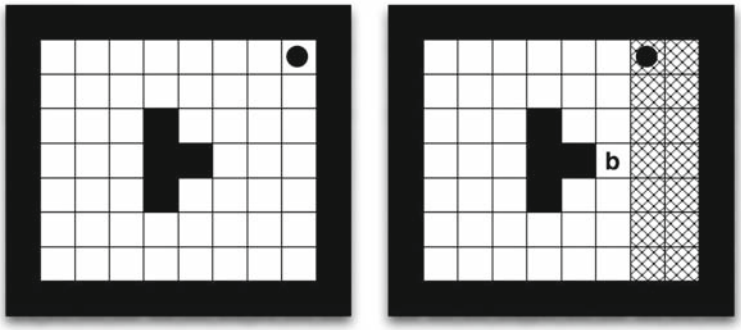

Fig. 8 The number of cyclic paths of accessible (*unexplored* or *explored*) cells around an obstacle gradually decreases as the agent marks more cells as *visited*. For example, the right map has fewer cyclic paths of white cells around the black cells than the left path. However, it is not possible to remove all cyclic paths by marking one of their common cells (e.g. cell *b*) as *visited*, as this would violate the marking rules of the Physical Agent Protocol, illustrated in Fig. 4

all cyclic paths of accessible cells around an obstacle, which means that it is impossible to achieve the *Termination Objective* in the presence of obstacles. An illustrative step-by-step example of this problem in a specific map scenario is provided in Fig. 9. □

Disadvantage 2 *If there are obstacles in the area, the Physical Agent Protocol is considerably slowed down in achieving the* Exploration Objective.

An example of the second disadvantage is shown in Fig. 10, where an agent is exploring the same loop several times, thus wasting time that could be employed in exploring unknown areas of the map. The cells in the loop (part L of the map) have been traversed only once. To go from the zone L toward the *unexplored* part of the map (U), the agent in the loop will need to explore every cell in it at least another 3 times, so that once traversing the cell pointed by the arrow, it will be able to choose the corridor on the left which leads toward zone U. Thus, before being able to escape the loop, the agent will need to traverse $3n$ cells, where n is the number of cells in the loop. The bigger the loop, the more time the agent will spend

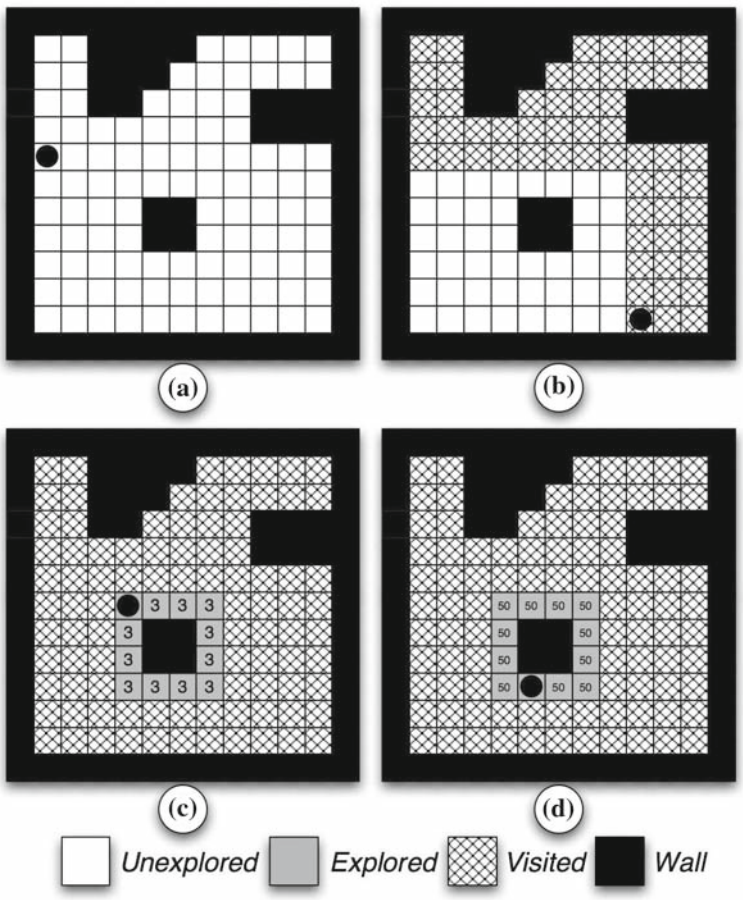

Fig. 9 A physical agent explores a room with an obstacle in the middle of it. At the beginning, all the cells in the room are *unexplored* (Stage **a**). While the agent explores the room, it thickens the walls with layers of *visited* cells (Stage **b**) until it forms a loop of *explored* cells around the obstacle (Stage **c**). To close the loop, the agent needs to mark one of the cells in it as *visited*. However, each cell is blocking the path between the two cells adjacent to it, and although an alternative path exists (the loop itself), the agent cannot know it locally. Therefore, no cell in the loop is marked as *visited*, and although the room has been fully *explored*, the agent will just keep moving along the corridor forever (Stages **c** and **d**)

in it, without being able to collaborate with the other agents in exploring other areas of the map. This waste of resources (agents) causes a longer overall exploration time, which is an essential parameter to be minimized in an emergency scenario.

To summarize, we have shown that whereas the Physical Agent Protocol always achieves the *Exploration Objective*, it takes considerably longer to do so in the presence of obstacles. In addition, it can never achieve the *Termination Objective* in the presence of obstacles. Both problems arise from the inability of the Physical Agent Protocol to cope with loops of accessible cells formed around obstacles. In the next section, we introduce a new protocol, called the Virtual Agent Protocol, which provides a loop-resolution mechanism, and works in synergy with the Physical Agent Protocol. Together the two protocols constitute the Hy-

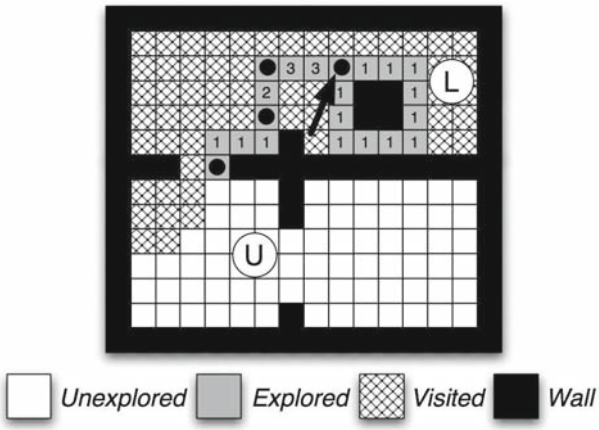

Fig. 10 The cells in the loop (part L of the map) have been traversed only once. To go from the zone L toward the *unexplored* part of the map (U), the agent in the loop will need to explore every cell in it at least other 3 times, so that once traversing the cell pointed by the arrow, it will be able to choose the corridor on the left which leads toward the zone U. This waste of resources (agents) causes a longer overall exploration time, which is an essential parameter to be minimized in an emergency scenario

bridExploration protocol that is capable of achieving both the *Exploration* and *Termination Objectives* in an efficient manner.

3.3 Virtual Agent Protocol

To speed up the exploration process and eventually achieve the *Termination Objective*, we introduce the concept of virtual agents. These are active messages propagated from cell to cell via the corresponding wireless tags. Virtual agents can only move to cells that are already deployed with tags; they cause changes in the current cell's state and make informed decisions about which cell to traverse next. Like physical agents, virtual agents move from one cell to an adjacent cell in the North, East, South or West direction, and they cannot traverse *visited* or *wall* cells. Unlike physical agents, they cannot traverse *unexplored* cells since there are no tags deployed there. Hence, they consider *explored* cells as their own territory, and build DFS trees along the corridors of *explored* cells that the physical agents leave behind. The goal of the virtual agents is to remove cyclic paths (loops) of *explored* cells.

The protocol that they run is a variant of the Depth-First-Search (DFS) algorithm. A detailed description of the Virtual Agent Protocol is provided in Algorithm 2. The main idea is that virtual agents extend the DFS tree with new *explored* cells in their way downwards, and mark these cells as *visited* when they traverse them in the opposite upward direction (as shown in lines 8–14 and 15–28 of Algorithm 2). An illustrative example of the Virtual Agent Protocol is provided in Fig. 11. The two agents first move together and include each *explored* cell in their way into the DFS tree (as shown in lines 4–7 of Algorithm 2). At the intersection, they continue to extend the DFS tree, but the one extends the left branch and the other the right branch. When the two virtual agents meet, they cannot extend the DFS tree any further; hence, they traverse the branches upwards marking cells in the way as *visited*.

The example above proved how the Virtual Agent Protocol tries to balance its resources across different branches of the DFS tree (one virtual agent followed the left branch and

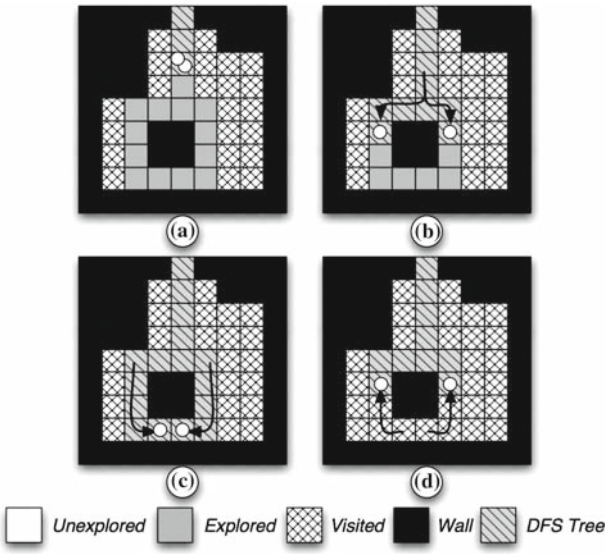

Fig. 11 Two virtual agents are collaborating to close a loop left by the physical agents. After the start (**a**), they separate into each of the two branches of the DFS tree (**b**) and meet again at the other side of the loop (**c**). Once there, they do not find any more *explored* cells which are not part of the DFS tree, so they close the loop by going back and marking every cell as *visited* (**d**)

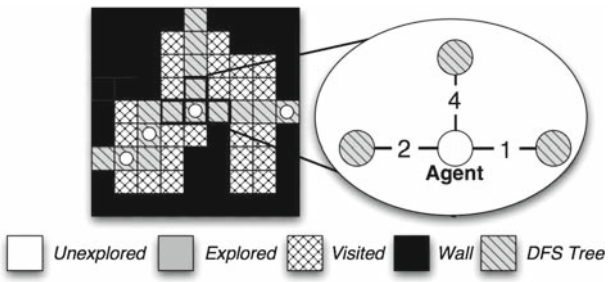

Fig. 12 In the DFS tree, each cell has four counters, associated with every edge connecting the current cell to the adjacent ones. The counter is updated by the agents and represents the number of agents which traversed the cell and went in the direction of that particular edge. The aim of the counters is to balance the agents during the exploration of the tree: at each node in fact, an agent will choose the edge which has been *explored* by the minimum number of other agents, so as to assign the same number of agents to each branch of the DFS tree. In the figure, the second virtual agent will go right, because the counter associated to that direction is 1 at the moment, and the previous agent has already updated the counter of the edge on the left from 1 to 2

the other the right branch). The mechanism used for distributing virtual agents to different parts of the network is as follows: each *explored* cell has a counter that measures how many virtual agents have traversed it coming from the same parent cell. Note that a cell can be accessed by a virtual agent from at most one parent cell. As virtual agents traverse the DFS tree downwards, they increment the counters associated with each traversed cell (as shown in lines 29–31 of Algorithm 2). At intersections, they choose to move to the *explored* cells traversed by the minimum number of virtual agents, so as to assign the same number of agents to each branch of the DFS tree. In Fig. 12, the second virtual agent will go right, because the

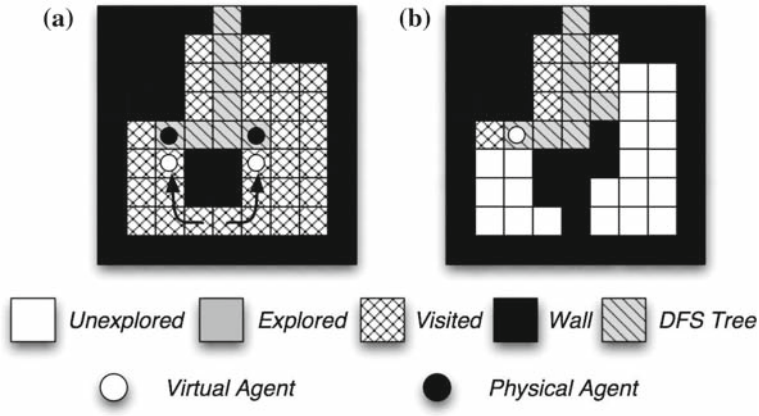

Fig. 13 Exploration rules for virtual agents. **a** Since the virtual agents are just messages, they move infinitely faster than the physical ones. To avoid virtual agents marking as *visited* a branch of the DFS tree in which a physical agent is exploring, the virtual agents always stop when they need to mark as *visited* a cell which is occupied by a physical one. As a result, a virtual agent could follow a physical one and mark the cells immediately behind it, literally "pushing" the physical agent toward *unexplored* areas. **b** When a virtual agent needs to mark as *visited* a cell with at least an *unexplored* adjacent neighbour, it stops until the unknown area is *explored* by a physical agent. In this way, *unexplored* areas will never be disconnected from the *explored* corridors

counter associated to that direction is 1 at the moment, whereas the counter associated with the left direction is 2.

We now discuss two rules that virtual agents should follow to work in harmony with physical agents. These rules are handled in lines 9–12 and 22–25 of Algorithm 2.

Rule 1: a virtual agent cannot mark a cell as *visited* if that cell is occupied by a physical agent, and it always has to wait until the physical agent is gone before continuing marking. An example of this behaviour is depicted in Fig. 13a, where virtual agents are following physical agents while they avoid "overtaking" them and subsequently blocking them in a branch with *visited* cells.

Rule 2: a virtual agent cannot mark a cell as *visited* if at least one of the adjacent cells is *unexplored*. An example of this case is provided in Fig. 13b, where the virtual agent stops any activity until the adjacent cell is *explored* by a physical agent.

To summarize, the Virtual Agent Protocol is capable of quickly removing cyclic paths of *explored* cells, formed by the Physical Agent Protocol. The role of virtual agents is to assist physical agents, by following them closely and cleaning up unwanted *explored* states in some of the cells, in order to help physical agents achieve the *Exploration* and *Termination Objectives* faster. The rules followed by virtual agents ensure that they always work in harmony with physical agents, i.e. they never delay or trap physical agents before the *Termination Objective* is achieved. For a detailed description of the Virtual Agent Protocol, the interested reader can refer to the pseudocode provided in Algorithm 2.

3.4 Analysis of the HybridExploration algorithm

Theorem 1 *The HybridExploration algorithm, which consists of the Physical Agent Protocol and the Virtual Agent Protocol, always achieves the* Exploration Objective.

```
   /* The cell in which the agent is located is defined as the current cell.                    */
   /* The cell from which the agent is coming is defined as the previous cell.                  */
   /* In the DFS tree the hierarchical relationship between the cells is defined as parent/child. */
   /* Every parent cell in the DFS tree has a counter for each of its children cells. The counter indicates
      the number of virtual agents which traversed the parent cell toward that particular child cell. */
 1 if you are coming from a cell which is child of the current one then
 2 |  decrease the counter of the current cell associated with the child;
 3 end
 4 if the current cell is not part of the DFS tree then
 5 |  mark the current cell as part of the DFS tree;
 6 |  define the previous cell as parent of the current one;
 7 end
 8 if the parent of the current cell is VISITED AND the current cell has one child cell only AND all the
   adjacent EXPLORED cells are part of the DFS tree then
 9 |  if there are any adjacent UNEXPLORED cells OR a physical agent is in the current cell then
10 |  |  stay in the current cell;
11 |  |  return
12 |  end
13 |  mark the current cell as VISITED;
14 end
15 if there are EXPLORED adjacent cells which are not part of the DFS tree then
16 |  go to one of them (each agent chooses a different cell according to its ID);
17 end
18 else if there is at least one child which is not VISITED then
19 |  go toward the child cell with the minimum associated counter;
20 end
21 else
22 |  if there are any adjacent UNEXPLORED cells OR a physical agent is in the current cell then
23 |  |  stay in the current cell;
24 |  |  return
25 |  end
26 |  mark the current cell as VISITED;
27 |  go to the parent cell;
28 end
29 if you are going to a cell which is child of the current cell then
30 |  increase the counter associated to that cell;
31 end
```

Algorithm 2 Virtual Agent Protocol of HybridExploration

Proof In Sect. 3.2, we proved that the Physical Agent Protocol alone always achieves the *Exploration Objective*, i.e. the physical agents eventually leave no cell in the *unexplored* state. The rules illustrated in Fig. 13 ensure that virtual agents do not block physical agents from achieving the *Exploration Objective*. The reasons are that virtual agents never disconnect two accessible parts of the network, and they never trap physical agents within *visited* cells away from accessible cells. □

Theorem 2 *The HybridExploration algorithm always achieves the* Termination Objective.

Proof When the *Exploration Objective* is achieved, all *explored* cells in the network are connected, since none of the Physical or Virtual Agent Protocols ever disconnect accessible parts of the network. Hence, the virtual agents will eventually finish building a single DFS tree over these *explored* cells and will eventually mark them as *visited* when traversing the branches of the tree upwards. □

4 Simulation results

We developed a simulation tool to test the performance of HybridExploration and compare it with three competing approaches: Ants [10], MDFS [14] and Brick & Mortar [14]. In fact, MDFS and Brick & Mortar are improved versions of the ones presented in [14]: MDFS agents are now capable of coordinating so that they are not trapped among *visited* cells; the loop resolution protocol of Brick & Mortar has been improved using timestamps and now is faster than the original version proposed in [14]. To introduce a further comparison with the existing literature, we also implemented the Ants algorithm presented by Svennebring et al. in [10]. The Ants algorithm performance appears in the graphs reporting only the exploration times, because this approach is not capable of achieving the *Termination Objective*. This happens because the Ants agents are not able to identify when the exploration terminates, therefore all cells remain permanently marked as *explored* and none of them is subsequently marked as *visited*. It is important to note that Ants has been devised with the aim of having computationally simple agents continuously sweeping an area, and therefore uses very basic rules, sacrificing initial exploration speed to the simplicity of the agents and the robustness of the system (e.g. Ants agents are immune to the kidnapped robot problem, because if one agent is moved from its location it can continue carrying on the exploration as nothing has happened). Our algorithm on the contrary focuses on a very rapid initial exploration and assumes slightly more capable agents in terms of computational power, thus the reader would probably need to consider this trade-off when looking at the plots in the present section. A detailed comparison between Ants and the original Brick & Mortar algorithm can be found in [14]. The developed tool allows us to automatically generate terrain maps with different topological features. Thus, we are able to study the impact of (i) the number of obstacles, (ii) the terrain size, (iii) the number of rooms and (iv) the number of agents on the performance of the three algorithms. Each point in the following graphs is the average of running an algorithm 20 times, each time with a different randomly generated map that satisfies the input topological features. In each experiment, we vary the values of one parameter, and assign default values to the remaining ones. The default values are: a map of 2500 (50×50) cells with 30 obstacles and 36 (6×6) rooms, which is explored by 20 agents. The agents are deployed from the top left cell of the area. We consider two performance metrics: (i) the *exploration time* (each graph on the left of the following figures), i.e. the number of steps required to achieve the *Exploration Objective*, and (ii) the *visiting time* (each graph on the right of the following figures), i.e. the number of steps required to achieve the *Termination Objective*. We chose the number of steps to evaluate our performance metrics because it represents the major source of energy consumption for the robot during the exploration process.

We also consider three different area types: (i) *Office*: area inspired by a real building plan, with corridors, offices, and an open-space area. (ii) *Collapsed*: area in which a severe event occurred (i.e. earthquake) and disrupted the normal plan of the building. (iii) *Series*: a long corridor which traverses several rooms; this scenario is inspired by a mine or another underground map in which each room has a door entering from the previous one and another door leading to the next. Examples of these area types are provided in Fig. 14. Our objective is to investigate how the performance of the proposed and competing algorithms varies depending on the spatial layout of rooms and doors in a building.

Effect of obstacles: Let us first study the impact of obstacles on the performance of HybridExploration and competing algorithms, as shown in Fig. 15. In all three scenarios, the introduction of obstacles does not seem to slow down MDFS towards achieving the

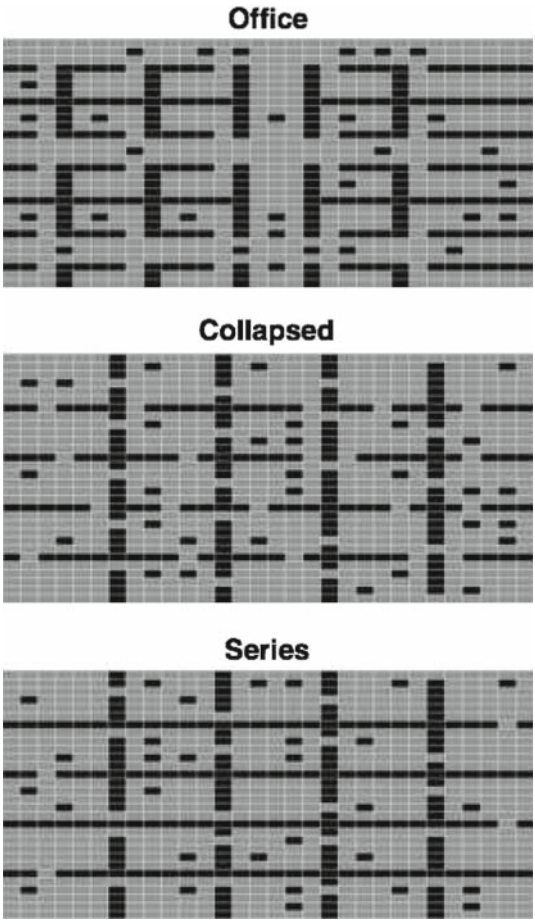

Fig. 14 Different scenarios in which the algorithms were tested

Exploration and *Termination Objectives*. In contrast, Brick & Mortar, which has a complex and time-consuming mechanism for resolving loops around obstacles [14], suffers from having to resolve an increasing number of obstacles. In all three scenarios, as one would expect, the plots denoting the exploration time of Brick & Mortar and MDFS cross over towards the middle of the x-axis, since Brick & Mortar is faster with few obstacles, but becomes inefficient with many obstacles. In the *Collapsed* scenario (Fig. 15c, d), we observe an unexpected behavior: the exploration time of Brick & Mortar is far lower than its visiting time. A careful study of this case revealed that Brick & Mortar agents were able to explore the area quite fast (thus achieving the *Exploration Objective*), but they interfered with each other whilst trying to resolve a large number of overlapping loops around obstacles. Thus, they required a much longer period to mark all cells as *visited*.

Our proposed algorithm, HybridExploration, has lower exploration and visiting times than the others in all three scenarios, and for varying numbers of obstacles. The reason is that it combines the ability of Brick & Mortar to quickly mark cells as *visited* when there are no obstacles, with the ability of MDFS to resolve loops, when there are many obstacles. It handles the presence of loops very well thanks to the virtual agents which are able to close them very quickly after they are created, while the physical agents are free to explore the

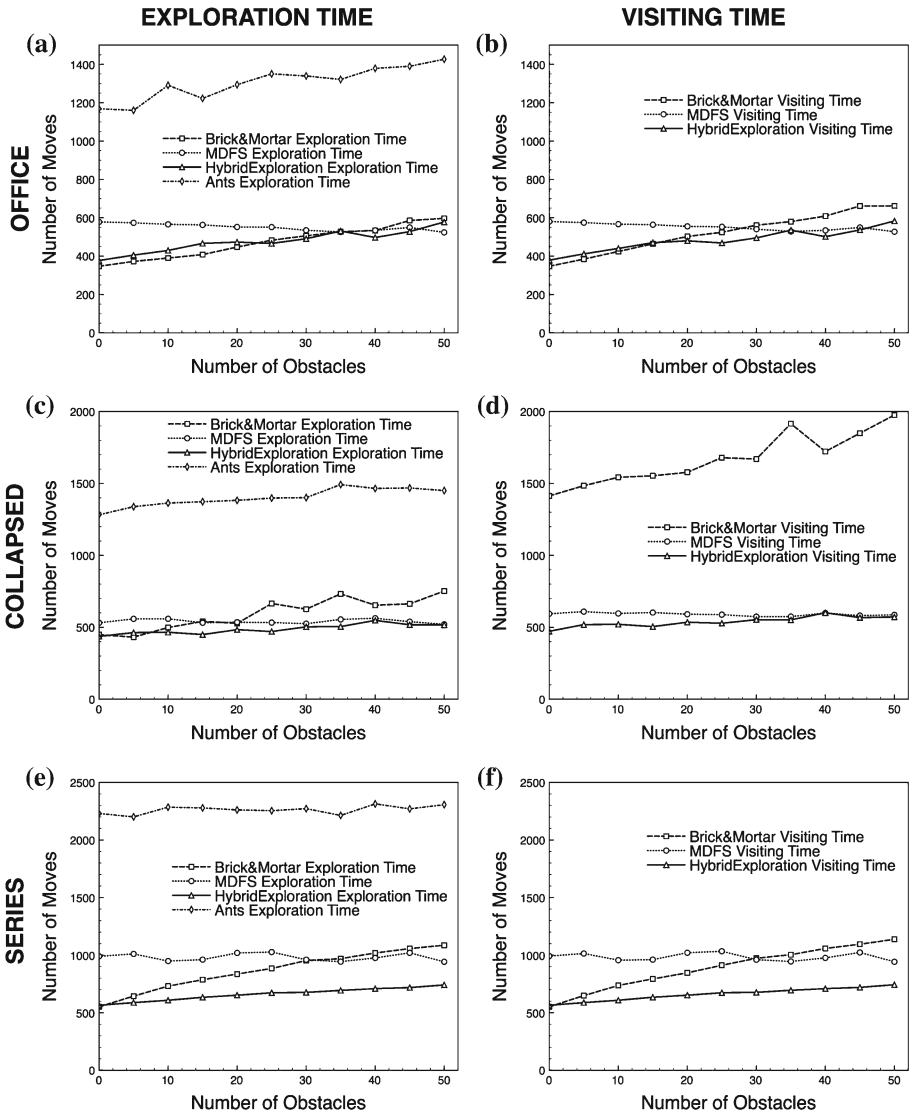

Fig. 15 Effect of changing the number of obstacles

remaining part of the area as fast as possible. The comparative benefits of HybridExploration are more pronounced in the Series scenario (Fig. 15e, f), where it is up to 50% faster than Brick & Mortar and MDFS.

Brick & Mortar is affected by obstacles, while MDFS can cope with them much more easily. HybridExploration, thanks to the virtual agent protocol, can maintain good performance even if the number of obstacles is increased.

Effect of area size: The next question that we address is whether the proposed algorithm scale gracefully as we increase the number of cells in the area. A comparison of the four algorithms in three different scenarios is depicted in Fig. 16. As one would expect, as we increase

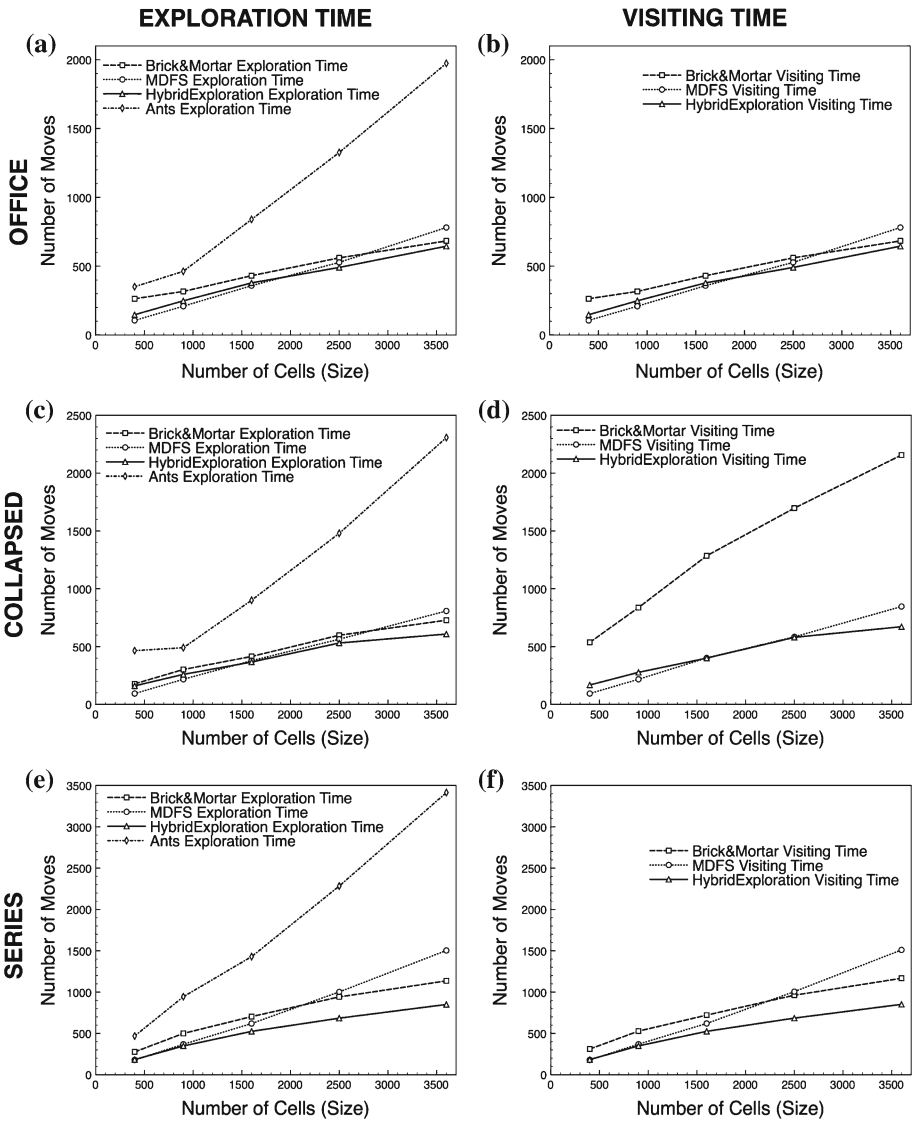

Fig. 16 Effect of changing the size of the map

the area size, the exploration and termination times increase for all three algorithms in all scenarios. Let us take a closer look at the behavior of MDFS and Brick & Mortar. In terms of exploration time, small areas favour MDFS over Brick & Mortar, whereas large areas favour MDFS. The two algorithms meet towards the middle of the x-axis in all three graphs. The reason is that MDFS typically traverses each cell more than once, whereas Brick & Mortar only once unless it has to resolve loops. The visiting times of the two algorithms is very close to the respective exploration times, except in the case of the *Collapsed* scenario where Brick & Mortar takes much longer to achieve the *Termination Objective* than to achieve the

Exploration Objective. The reason for this gap is explained above in our discussion of the effect of obstacles.

Our HybridExploration algorithm, which combines the strengths of MDFS and Brick & Mortar, outperforms both of them in all three scenarios, and scales gracefully with the area size. In the Series scenario (Fig. 16e, f), where the loops around obstacles are longer than in the other two scenarios, the ability of HybridExploration to close loops using virtual instead of physical agents gives it a significant advantage over Brick & Mortar and MDFS.

HybridExploration scales gracefully with the number of cells, thanks to the positive effect of the Physical agent protocol.

Effect of rooms: Figure 17 shows the effect of varying the number of rooms in the area while maintaining the same number of cells, and thus the same area size. It is peculiar how Brick & Mortar performs poorly in terms of visiting time, when there are no rooms at all (open space area). This can be explained by observing that the loops are not bound within a room but the agents can build loops which are as large as the whole area, and therefore the time needed to close them is much longer than in cases where a loop cannot be larger than the size of a room. In the *Collapsed* scenario (Fig. 17c, d), however, an increase in the number of rooms slows down Brick & Mortar. The reason is that in this particular scenario, rooms have many collapsed walls (doors), and more rooms effectively introduce more loops that are difficult and time-consuming to resolve.

The behavior of MDFS is also interestingly different depending on the scenario. In the *Office* and *Collapsed* scenarios (Fig. 17a–d), the performance of MDFS both in terms of exploring and visiting times hardly depends on the number of rooms. We observe a small decrease in the exploration and visiting times of MDFS, simply because some of the cells that were previously *unexplored* are now *wall* cells forming the frames of rooms; these *wall* cells do not require to be traversed and marked as *explored* or *visited*. The behavior of MDFS in the *Series* scenario (Fig. 17e, f) is rather unexpected. MDFS is slowed down by the presence of rooms although with more rooms there are fewer cells to cover. We think that this is because one of the strengths of the MDFS algorithm is that the agents can build trees of *explored* cells, with several branches spanning different rooms in the scenario, and process them at the same time. This parallelisation though is not possible in the *Series* scenario, where the rooms form a chain and need to be explored one after the other, without the possibility to explore more than one at the same time. The agents running MDFS therefore need firstly to mark the cells in the room as *explored*, then traverse them again to mark them as *visited* and finally move on to the next room, without efficiently using all the agents in a parallel way during the exploration. The Brick & Mortar and HybridExploration algorithms on the contrary can directly mark every cell as *visited*, so that, although they cannot explore more rooms at the same time in parallel, the overall exploration and visiting times are much less than the MDFS ones.

The HybridExploration algorithm outperforms the other two algorithms in both scenarios; again, the Series scenario presents the most interesting case, where the benefits of HybridExploration are more pronounced.

Effect of agents: Figure 18 shows the effect of varying the number of agents exploring the area. From the graphs, we notice how the performance of all algorithms in terms of both exploring and visiting times is improved as more agents are used but up to a certain threshold. After this threshold is achieved, using more agents does not lead to better exploration or visiting times. This happens because there is a limit to the parallelism of the exploration that the agents can achieve, determined by the size of both rooms and the whole exploration area. Notice that HybridExploration always achieves better results than the other algorithms when relatively few agents (i.e. less than 30) are used. Furthermore, even when the number

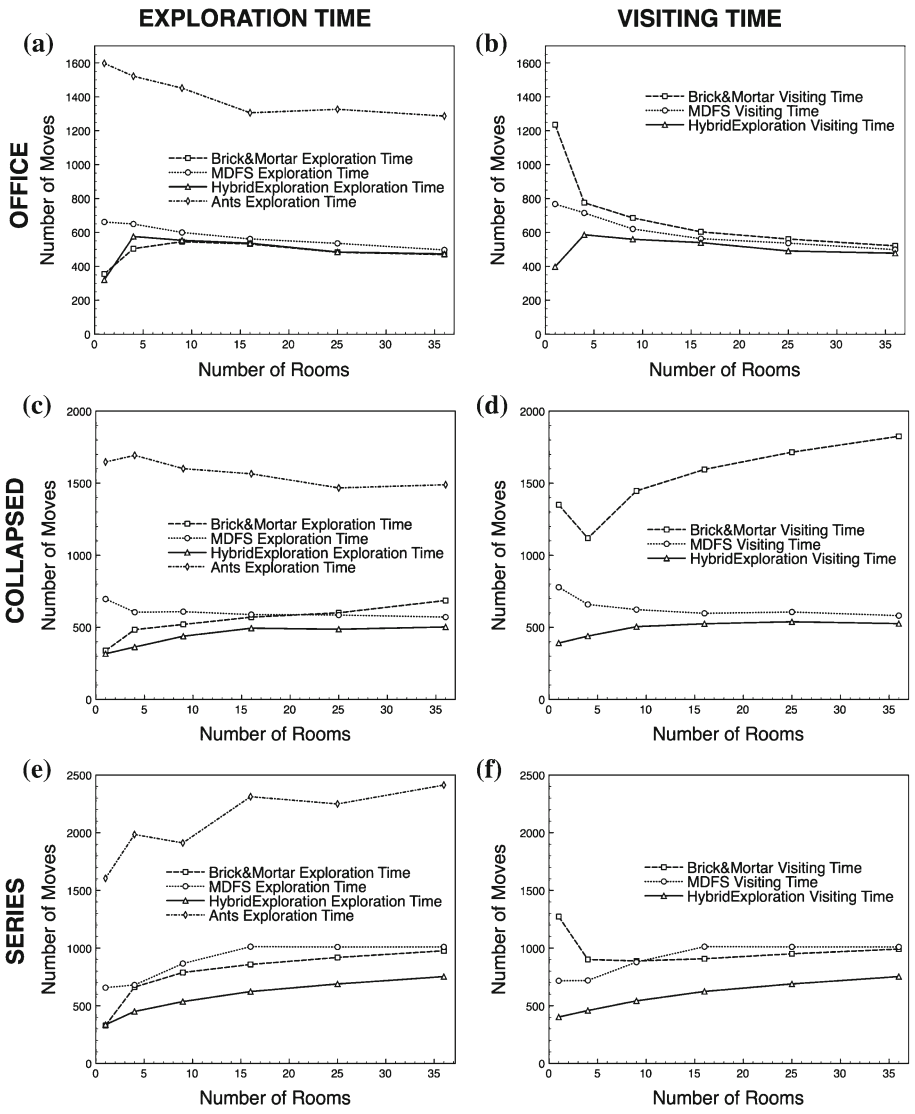

Fig. 17 Effect of changing the number of rooms

of agents is increased above that number, the performance of HybridExploration remains comparable to that of the others. Once again, the benefits of HybridExploration over the competing algorithms are more pronounced in the *Series* scenario.

As expected, the performance of HybridExploration and competing algorithms improves as we increase the number of collaborating agents. Initially, when the number of agents is low, the benefit of adding one more agent is significant for all algorithms. However, there exists a threshold of agents beyond which no significant performance improvements are detected. This threshold tends to be lower for HybridExploration than for competing approaches.

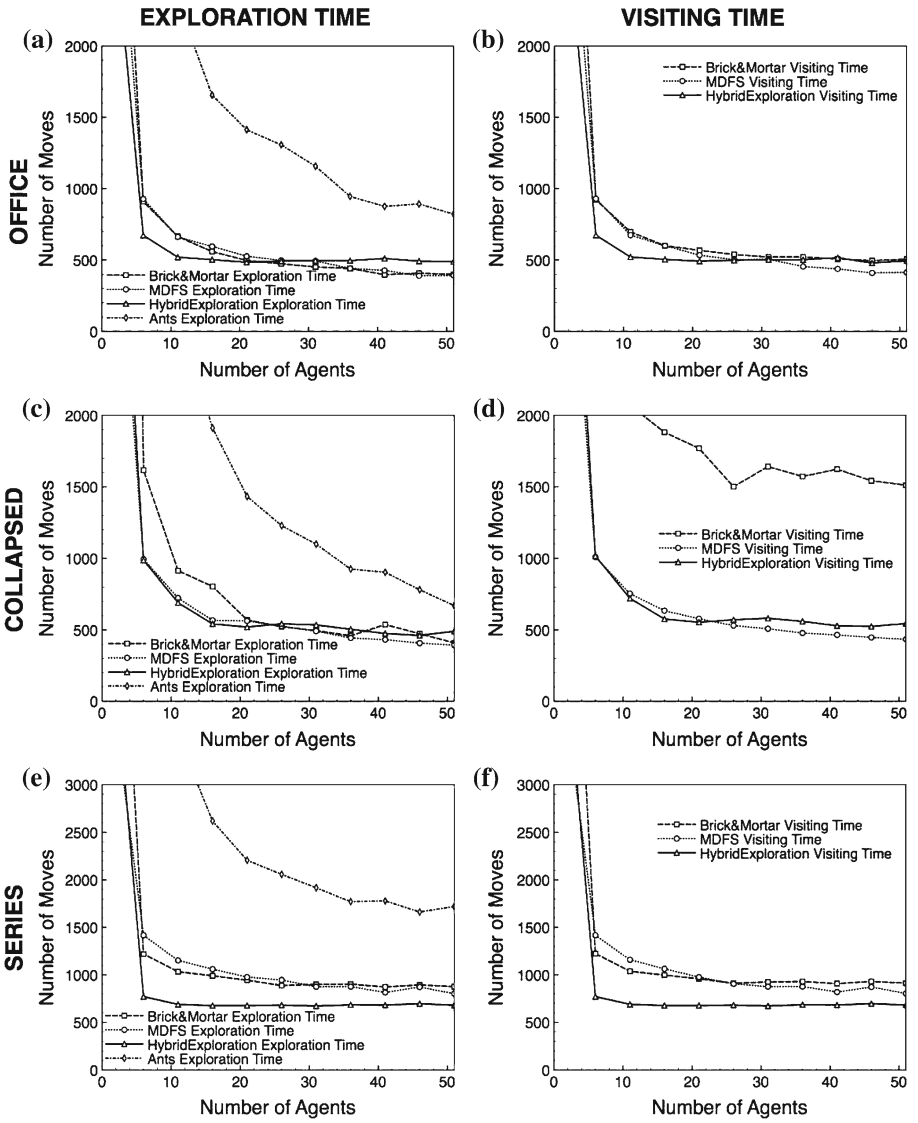

Fig. 18 Effect of changing the number of agents

HybridExploration performs similarly to the best of competing approaches in the Office and Collapsed scenarios, and is clearly superior in the Series scenario.

Summary of results: The HybridExploration algorithm outperforms the three competing approaches (Ants, MDFS, and Brick & Mortar) in terms of exploration and visiting time, in all three scenarios (*Office*, *Collapsed*, and *Series*), and for a wide range of area sizes, obstacle, room and agent numbers. The benefits of HybridExploration compared to Ants, MDFS and Brick & Mortar are far more obvious in the *Series* scenario. In general, HybridExploration is faster because it combines the strengths of all the competing approaches. First, like MDFS it adopts a depth-first-search approach to resolve loops around obstacles, but using virtual

agents instead of physical agents. Second, like Brick & Mortar, agents running HybridExploration traverse each cell approximately once before they mark it as *visited*. Finally, a variation of the Ants algorithm is used to let the physical agents escape from loops. Interestingly, whereas Brick & Mortar tends to traverse cells multiple times when we introduce obstacles, HybridExploration is more robust when obstacles are present, because it resolves loops around them much faster with the aid of virtual agents.

5 Related work

Information gathering inside an area is essential to avoid risking the lives of the first responders: for example, if the responders knew the locations of the victims before entering a building, they could immediately get there avoiding hazardous areas such as rooms on fire or collapsed corridors or stairs. Exploring all the area in the minimum amount of time and reporting back to the human personnel outside the building is therefore an essential part of rescue operations. A group of mobile agents should therefore be deployed in the area to acquire all the information that could assist the tasks of the first responders. The existing algorithms used by the agents to perform the exploration task can be distinguished based on two criteria: (i) knowledge of map and (ii) communication modes.

1. *Knowledge of map*: Choset [16] provides a survey of coverage algorithms and distinguishes them into *off-line* and *on-line*. In the former, the agents are previously provided with a map of the area to explore, while in the latter, also called *sensor-based*, no assumption is made concerning the availability of an environmental map for the agents.
2. *Communication modes*: the robots communication can be classified in three main different ways:
   - *Agent-to-Server communication*: the robots coordination occurs through a central server;
   - *Agent-to-Agent communication*: a direct wireless communication occurs between robots within the same range;
   - *Agent-to-Environment communication*: agents communicate indirectly by interacting with an instrumented (smart) environment.

Our approach differs from related work in that we are investigating the subclass of *on-line* algorithms that rely on communication through the instrumented environment (Agent-to-Environment). Usually in fact the *on-line* approaches assume that the agents are able to coordinate their movements using long range radio frequency (RF) communication. However, in the literature [17–23] it is widely accepted that radio propagation is (i) non-isotropic (i.e. the received signal, at a given distance from the sender, is not the same in all directions), it has (ii) non-monotonic distance decay (i.e. lower distance does not mean better link quality), and (iii) the communication is based on asymmetrical links (i.e. if A hears B, it cannot be assumed that B hears A). For this reason, the agents coordination through RF communication represents an assumption that is not always true.

The unreliability of the wireless communications leads us to explore a relatively novel class of algorithms within the *on-line* paradigm, in which the agents are able to tag the environment and update the state of the tags in order to communicate and coordinate with each other, without relying on long range RF communication. In particular, we envision the use of inexpensive tags that can be left on the ground and which the agents can use to store and retrieve data (Agent-to-Environment). The feasibility of the approach is supported by Hähnel

et al. [24], who proved how a robot can use RFID tags already placed on an area to localize itself and navigate through the rooms, and recently by Kleiner et al. [25], who presented a robot which is able to autonomously drop RFID tags on the environment and implement an existing *on-line* exploration algorithm [26].

To the best of our knowledge, little work [10,14,15,25] has investigated the problem of *on-line* area exploration by letting agents coordinate indirectly by tagging the environment, subsequently reading and updating the state of the deployed tags. Moreover, only few of those algorithms are able to autonomously decide when the exploration is terminated. Recognising when the exploration terminates is in fact of primary importance in an emergency scenario such as the one we are tackling: if the robots have to report back to the first responders the situation inside the building, they absolutely need to know when to stop exploring, and to do that they need to be sure whether all the area has been explored or not. In our previous work [14], we proposed two algorithms, MDFS and Brick & Mortar, to solve the terminating issue in a distributed fashion by marking cells already visited by the agents, but these algorithms do not exploit tag-to-tag communication, i.e. they do not use the multi-hop communication capabilities of the deployed sensor network. Furthermore, agents running MDFS are poorly coordinated, and this leads toward long exploration times, while agents running Brick & Mortar use a complex loop resolution mechanism that significantly delays the task of area exploration, especially in topologies with many obstacles (e.g. desks in the middle of an open space). Our proposed approach, HybridEploration, has been designed to address all these weaknesses and achieve even better performance by making advantage of tag-to-tag communication.

The following subsections will provide detailed descriptions of the state of the art in the field of exploration algorithms. The approaches most related to our work will be presented first, namely Ants (Sect. 5.1), MDFS (Sect. 5.2) and Brick & Mortar (Sect. 5.3). Following that, a more general survey of on-line algorithms will be presented in Sect. 5.4, while off-line approaches will be described in Sect. 5.5. Finally, related to our work is also the SLAM exploration, to which Sect. 5.6 is dedicated.

5.1 The Ants algorithm

In this section, we first discuss the behaviour, strengths and limitations of the Ants algorithm proposed by Svennebring and Koenig in [10,15]. This is a distributed algorithm that simulates a colony of ants leaving pheromone traces as they move in their environment. Initially, all cells are marked with value 0 to denote that they are *unexplored*. At each step, an agent reads the values of the four cells around it and chooses to step onto the least traversed cell (the one with the minimum value). Before moving there, it updates the value of the current cell, for example by incrementing its value by one. The authors discuss a few other rules that could be used instead to mark a cell and navigate to the next one, but they all exhibit similar performance in terms of exploration time. Hence, we select the above variant of the Ants algorithm (move to the least *visited* cell) as a basis for comparison. The authors provide a proof that the agents will eventually cover the entire terrain (provided that it is not disconnected by *wall* cells), and thus that the *Exploration Objective* is always achieved.

The first advantage of the algorithm is its simplicity: agents do not require memory or radio communication, but only one-cell lookahead. Since they are easy to build, many of them can be used to shorten the coverage process. Secondly, there is no map stored inside the agents: if one of them is relocated (accidentally or on purpose) it will not even realise it

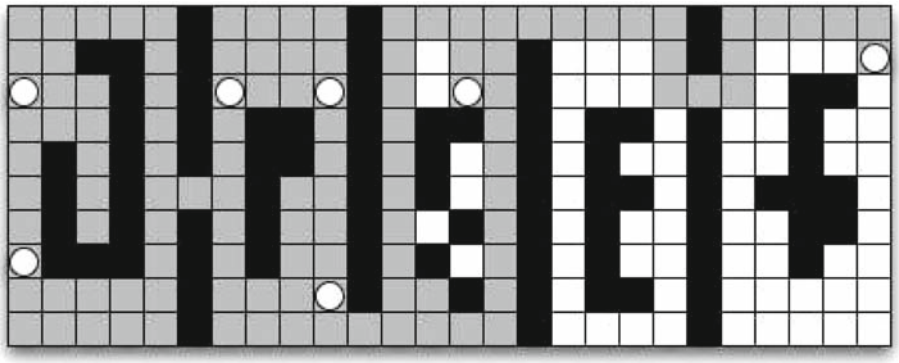

Fig. 19 The Ants algorithm is not efficient in a scenario with many rooms, because most of the agents explore the first rooms repeatedly, while only few of them set out to discover new areas

and it will continue to do its work as if nothing happened. This means that the whole system is flexible and fault tolerant, and the area can be covered even if some markings or agents are lost. At the storage device of each cell, we only need to store an integer counting the number of times that agents have visited the cell. When the number of times exceeds a threshold, the counter is reset to 0.

The main limitation of the Ants algorithm is that the *visited* state is not used during the process of marking cells. Therefore, the *Termination Objective* is never achieved, and the agents continue the exploration phase until they run out of energy.

Thus, this approach is not suitable in an emergency scenario, in which the primary consideration is to cover the overall area as soon as possible, and be notified immediately after the task is completed. A further drawback of the algorithm is the limited collaboration among agents. As shown in Fig. 19, in a scenario with many rooms most of the agents would sweep the first few rooms repeatedly, while only a few of them would venture to explore new areas, thus limiting the efficiency of the algorithm when using multiple agents.

5.2 The multiple depth first search algorithm

In order to address the limitations of the Ants algorithm detailed in Sect. 5.1 (i.e. lack of collaboration among the agents, slow achievement of the *Exploration Objective* and no achievement of the *Termination Objective*), a Depth First Search (DFS) approach has been proposed in [14]. Unlike Ants, this algorithm allows agents to mark cells as *visited*, so that agents do not need to traverse them in the future. As a result, an agent knows that its task is completed if its four adjacent cells are either *visited* or *wall* cells. The values used to annotate cells by the Ants algorithm are not used in this case.

We first consider the case with a single agent. The agent explores the area by moving to the next *unexplored* cell, marking it as *explored*, and storing in it the direction of the previous cell (i.e. North, East, South, West). In doing so, it builds an exploration tree in which each cell has a parent cell (the cell where the agent came from, before moving to the current cell for the first time). When there are no *unexplored* cells adjacent to the current cell, the agent has reached the end of a branch and is ready to start traversing it backwards. It marks the current cell as *visited* and moves to the parent cell. This is repeated until the agent finds an adjacent *unexplored* cell and moves to it to start marking a new branch as *explored*. In short,

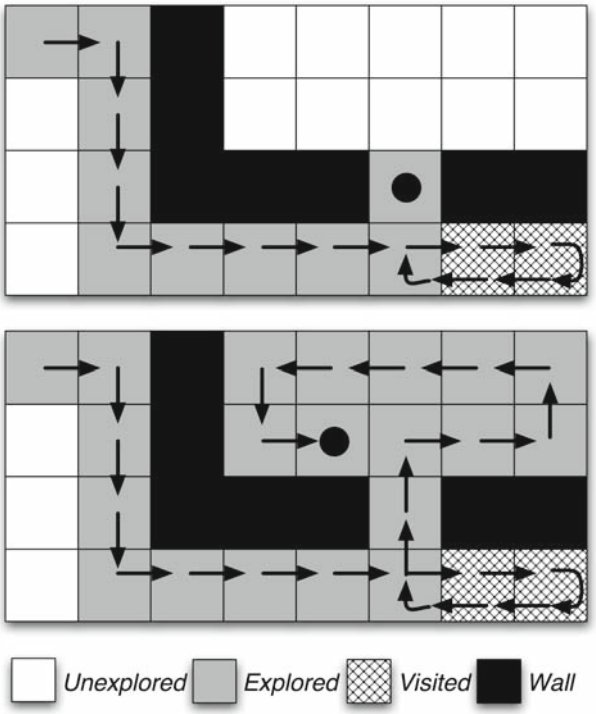

Fig. 20 DFS exploration example

most cells (except for leaf cells) are traversed at least twice, once marked as *explored*, as the agent traverses the branch downwards, and once marked as *visited*, as the agent traverses the branch upwards. When the agent is back at the root cell and all adjacent cells are either *visited* or *walls*, the algorithm terminates.

A snapshot of the exploration process with one agent can be seen in Fig. 20: the agent starts at the cell in the top left corner of the area, and decides to move on the path denoted by the arrows, annotating cells in the way as *explored*. When it reaches the cell at the bottom right corner, it is surrounded by either *wall* or *explored* cells, and realises that it is at the end of a branch. It starts moving backwards marking the cells of the branch as *visited* until it identifies the start of a new branch. The first cell of the new branch is the one between the two *wall* cells, which is initially *unexplored*. It starts processing the second branch by marking that cell as *explored* and repeating this step as it traverses the branch downwards, until it reaches the current position denoted by the black sphere. At this point, it identifies the end of another branch, and it will start traversing the branch upwards and marking its cells as *visited*. The agent will continue the exploration task in a similar manner until it is surrounded only by *visited* or *wall* cells.

The challenge in using more than one agent is to ensure that they can efficiently collaborate to explore the area. In the extended Multiple Depth First Search (MDFS) algorithm, the protocol that the agents run is a variant of the Depth-First-Search (DFS) one.

The main idea is that agents extend the DFS tree with new *explored* cells in their way downwards, and mark these cells as *visited* when they traverse them in the opposite upward direction. In the DFS tree the hierarchical relationship between the cells is defined as

parent/child. Moreover, the mechanism used for distributing agents to different parts of the network is as follows: each *explored* cell has a counter that measures how many agents have traversed it coming from the same parent cell. Note that a cell can be accessed by an agent from at most one parent cell. As agents traverse the DFS tree downwards, they increment the counters associated with each traversed cell. At intersections, they choose to move to the *explored* cells traversed by the minimum number of agents, so as to assign the same number of agents to each branch of the DFS tree.

Using this algorithm the agents are typically able to explore the area in less time than using the Ants algorithm, and more importantly, each agent knows exactly when to stop its exploration task, and thus the algorithm is capable of achieving both the *Exploration* and *Termination Objectives*, as described in [14]. Hence, the algorithm terminates when all agents stop moving, since when that happens all cells are marked as *visited* or *walls*. Although the Multiple Depth First Search addresses some of the weaknesses of the Ants algorithm, it is still not very efficient in terms of exploration time. By definition it traverses each cell at least twice (except for leaf cells), thus resulting in a long exploration time even in open areas without walls where a single traversal would suffice. Furthermore, the approach does not make use of the tag-to-tag communication, which could be exploited to better coordinate the agents.

5.3 The Brick & Mortar algorithm

The Brick & Mortar [14] algorithm is designed to address the weaknesses of the two algorithms seen so far in Sects. 5.1 and 5.2. Unlike the Ants algorithm, agents using Brick & Mortar know when the exploration task is completed and they do not spend much time revisiting the same cells. Unlike MDFS, agents typically traverse each cell less than twice, thus resulting in a shorter exploration time.

The driving idea is that of thickening the existing walls by progressively marking the cells that surround them as *visited*. Once again, *visited* cells are equivalent to *wall* cells in that they can no longer be accessed. In the description of the algorithm, we refer to *wall* and *visited* cells as inaccessible cells, and to *unexplored* or *explored* cells as accessible cells. The algorithm aims to progressively thicken the blocks of inaccessible cells, whilst always keeping accessible cells connected. The latter can be achieved by maintaining corridors of *explored* cells that connect all *unexplored* parts of the network.

Like Ants and MDFS, Brick & Mortar does not require agents to know their location in the building. A relocated agent can simply navigate randomly until it finds an accessible cell and then continues the exploration from there. Brick & Mortar makes the blocks of inaccessible cells thicker until the entire terrain is converted to a large block of inaccessible cells. In a rectangular terrain without *wall* cells, agents starting from border cells always succeed in visiting the entire area. In more complex topologies with many rooms and obstacles, agents may be faced with a loop closure problem. In particular, if there are obstacles in the area, the Brick & Mortar algorithm without loop closure does not always achieve the *Exploration Objective* and never achieves the *Termination Objective*. On the other hand, with a loop closure procedure, the Brick & Mortar algorithm always manages to achieve both the *Exploration* and the *Termination Objectives*. However, the loop resolution mechanism is complex and often introduces delays in the exploration when too many obstacles are found, and the algorithm does not use tag-to-tag communication to coordinated the agents.

5.4 On-line algorithms

On-line algorithms should rely only on their sensors in order to navigate an unknown environment and be capable of taking *on-line* decisions about what to do next. Having many different possible environments, one algorithm could work better in an indoor environment with tiny rooms and a great number of corridors, while another could be faster in the case of big rooms interconnected by many doors, and so on. Most of these approaches divide the environment into cells, also called regions, that are explored one by one iteratively until the global area is covered. Furthermore, on-line algorithms are comparable to our approach with regard to hardware cost and complexity, since they all make use of small inexpensive robots with few sensors, and are bounded in their exploration time [9].

A very important contribution can be found in the work of Wagner et al. [9,27–29], who extensively covered the use of small inexpensive robots (A(ge)nts) to explore an unknown area. They formalized the exploration problem in a more analytical way (a one-robot formalization is also available by Albers et al. [30]), proved that the off-line coverage problem is NP-Hard, and proposed very interesting heuristics to solve the on-line version. Furthermore, in their most recent work [9] they envisioned an approach (CLEAN) presenting several similarities with the Brick & Mortar algorithm [14], which is also at the core of the HybridExploration solution presented in this paper. In particular, robots make use of the status of 'dirtiness' of floor tiles to determine if someone already explored a certain area, and clean a tile to "mark" it as already visited. However, this method is incapable of solving loops when scenarios with obstacles are considered; to address this problem, the authors suggest a loop resolution mechanism (using additional markings) that is similar to the one used by Brick & Mortar [14]. Furthermore, since only two "status" are used (dirt or not) their algorithm would be less efficient than Brick & Mortar during the exploration. In fact, agents running Brick & Mortar can choose to explore *unexplored* cells preferring them over *explored* ones, therefore speeding up the exploration time. For this reason, we chose to compare HybridExploration with Brick & Mortar (instead of CLEAN) in the experimental section.

Batalin and Sukhatme have done plenty of work [1–4] using radio beacons to guide the navigation of robots and assist them in the coverage of an unknown terrain. In particular, their robots are able to detect the beacons (which are pre-deployed into the environment), choose one of them and move toward it, always guided by their radios. The beacons are able to tell the robot on which direction (North, West, South or East) the least recently visited neighbour beacon lies. Beacons and robots are both equipped with a 2-bit compass, so the former can give the latter indications about which direction to take to reach the next beacon. Furthermore, in [31], the same authors suggest that the beacons could be dynamically deployed by a robot (an approach similar to our previous work [14]), which could then use the same Least Recently Visited (LRV) navigation approach. We believe that the relatively simple algorithm they use could be further improved to reach better performance in exploring the area as fast as possible.

The use of a pre-deployed embedded network to assist the navigation of a robot in the environment has been extensively studied by O'Hara et al. [5–8]. In particular, the authors use small and relatively inexpensive embedded network platforms, the GNATs, to guide the navigation of a LEGO Mindstorm/RCX robot using infrared transmitter and receivers. The main strength of this work relies on the extensive real-world experiments that the authors have done (up to 156 nodes deployed), which proves the feasibility of their approach and of the Agent-to-Environment communication paradigm in general.

Finally, Li et al. [32,33] also use a sensor network to help a user (human or robot) to safely traverse an hostile environment, but the positions of the nodes are always known (by

using a GPS on each of them) and in the real experiment there is no communication between the user and the network (information from the LEDs of the nodes are video recorded and examined a posteriori).

A different approach (following an Agent-to-Server communication mode) has been presented in [34]. In this work, the authors use a Boustrophedon [35] technique to let the robots cover the area. The environment is divided into cells and each robot starts covering each cell with forward and reverse phases. While doing this, it builds a graph of the environment which is shared by all the robots of the team. Due to this fact, the robots always know where there are uncovered cells and therefore they can distribute themselves over the area in order to cover the remaining unexplored regions. The proposed algorithm is globally simple and easy to implement on real hardware. It also leads toward very efficient coverage time because the robots always know where the unexplored zones are, so they can go directly on them without wasting time in idle states. The graph is shared among all the agents so that, if one of them has a fault, the covering procedure can continue without losing any information. However the assumptions are different from the ones we are adopting for the present work, in particular our agents do not rely on perfect wireless communication among them and we do not assume that they always know where they are in the map (perfect localization).

Yamauchi presents a frontier-based exploration algorithm [36], where the agents explore the environment, represented by a regular grid of cells, keeping in their memory a map of the area and always directing themselves "to the boundary between open space and uncharted territory". A depth first search algorithm is used to move from the current position to the next frontier. Each agent has a local map and a global map shared with all the other agents. When a local map is updated (the agent explores a new area), it is summed with the global map, and the latter is broadcasted to all the other agents so that they can update their global maps. The agents do not broadcast information about what area will be explored next, so different agents could explore the same frontiers in the same time resulting in an inefficient utilization of the team. The algorithm also makes the same strong assumptions that we found in [34] namely perfect localization, communication and mapping, and uses an Agent-to-Server communication approach, because the map is stored in a centralized server.

Another Agent-to-Server communication approach is presented by Howard et al. [37], where the authors show a general approach to explore a building, find objectives, and report them back to the human personnel outside. However, it requires human support to solve problems like loop closures or map merging between the agents, so it does not satisfy the requirements for autonomous area coverage.

Finally, works which use the Agent-to-Agent communication paradigm have been proposed by the following authors.

Burgard et al. [38] do not assume that the environment is divided into grid cells. Agents compute an utility function to go to the next "frontier" in order to maximize the explored territory. Although the agents have to store information about the map and localize themselves, this approach is probability-based and therefore more suitable for a real scenario: the agents have a list of target points to reach in the next step, each of them associated to a value which takes into account the cost to reach the point, and the probability of exploring new areas once an agent has positioned itself on that point. The probability takes into account how many agents are going to explore the area in which the target point is, therefore avoiding the situation where all the agents explore the same area. Rekleitis et al. [11] try to improve the exploration by mapping the environment while the area is covered by two robots. To localize the robots they use odometry (a position estimate based on the previous movements), but while a robot is exploring the area, the other stands still and observes the former to measure its movements and improve the localization; after a certain amount of time the two robots

exchange roles. The authors map the environment as trapezoids divided into striped which are connected forming a graph. The agents use a depth first search algorithm to cover all the stripes.

Batalin et al. [39], who focus on agent dispersion and propose two algorithms to make the agents move away from each other when they are in sensing range. This is an important issue because the agents should always be spread out as much as possible in the environment to avoid wasting resources in exploring the same area multiple times.

Another interesting approach, inspired by ants behaviour, is proposed by Payton et al. [40]. The authors use a swarm of robots which spread into the environment, and coordinate between themselves through infrared communication. Messages are propagated through the robots (which act both as explorers and as relays for messages) to send information about the distance and position of a goal that must be reached.

5.5 Off-line algorithms

Since the *off-line* algorithms previously know the map of the environment, in theory they could build a set of optimal paths to cover the area using all the available agents in the minimum amount of time (optimal solution). In practice, this is not possible because this problem is NP-complete, therefore heuristics are needed to find a feasible solution in polynomial time. Such a heuristic is proposed in [41], where the authors prove that the original problem is NP-complete, and propose a polynomial algorithm, which runs in the worst case eight times slower than the optimal solution. Agmon et al. [42] propose a faster tree construction algorithm, while Hazon et al. [13,43] focus on the robustness of the solution, so that even if only one robot remains in operation, it will be able to carry and complete the exploration task. All the *off-line* algorithms, being centralized in their computation of the exploration paths, use the Agent-to-Server communication mode.

5.6 Simultaneous Localization and Mapping (SLAM)

In this section, we provide a brief overview of the Simultaneous Localisation and Mapping (SLAM) technique. Our aim is to describe its main characteristics while emphasising the reasons that make it different from our approach. A detailed description of SLAM can be found in two tutorials by Hugh Durrant-Whyte and Tim Bailey [44] and [45]. The SLAM problem investigates the possibility for a mobile robot to be placed at an unknown location in an unknown environment and to incrementally build a map of the environment while simultaneously determining its position within the map. In particular, the SLAM is a process by which a mobile robot can build a map of an environment and at the same time use this map to deduce its location. In SLAM, both the trajectory of the robot and the location of all landmarks are estimated online without the need for any a priori knowledge. The robots just use, without actively modifying, the information extracted from the environment both to estimate their position and to dynamically build a map of an unknown area. On the other hand, the aim of our work focus on the ability of a team of autonomous robots to collaborate in the task of exploring an unknown and hazardous terrain in the minimum amount of time. Fast exploration and coverage of unknown areas, along with coordination of the agents, are thus the primary concerns of our work. Moreover, our approach actively modifies the environment tagging it with devices able to store information during the exploration process. Finally, our approach does not use observations to estimate robot locations.

6 Conclusion and future work

In this work, we proposed a new algorithm, HybridExploration, for the multi-agent exploration of unknown terrains. Our algorithm is based on an architecture consisting of both mobile nodes (robots, called *agents*), and stationary nodes (inexpensive smart devices, called *tags*). As the former enter the emergency area, they drop tags into the environment to label it with a *state*. By reading and updating the state of the local tags, agents are able to coordinate indirectly with each other. In addition, agents are further assisted during the exploration task by tags able to wirelessly exchange local information. We compared our novel approach against three existing algorithms, Ants, Brick & Mortar, and Multiple Depth First Search. Our algorithm avoids exploring the same areas multiple times, it makes good utilization of both physical and virtual agents and it is capable of efficiently resolving loops. Our simulation results show that our HybridExploration algorithm, combining both tag-to-tag and agent-to-tag communication, is significantly faster than the three competing ones, Ants, Brick & Mortar, and Multiple Depth First Search, in a variety of scenarios.

As future research, we would like to explore the use of tags to create a network infrastructure, in order to communicate interesting events back to human responders and assist them in the task of finding the cells where events were detected. Such an infrastructure could also be used to monitor the surrounding environment, so that if an adjacent tag is malfunctioning, or a section of a corridor collapses forming a new physical obstacle, the information could be propagated immediately and alternative paths could be discovered by virtual agents, using the same exploration algorithm. In this way, we could both monitor the explored areas and cope with a dynamic environment. Another challenge is to cope with different cell distributions, without assuming that the tags are deployed in a grid network. In particular, we plan to extend the proposed approach to the general case of connected graphs of cells. Furthermore, we would like to deal with communication failures in tag-to-tag and agent-to-tag communications. Finally, we would like to provide a formal termination proof for the proposed algorithm.

Acknowledgments Effort sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-06-1-3003. The U.S. Government is authorized to reproduce and distribute reprints for Government purpose notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government. The authors would like to thank the Reviewers and the Editor for their useful comments.

References

1. Batalin, M. A., & Sukhatme, G. S. (2005). The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment. In *ICRA05: Proceedings of 2005 IEEE International Conference on Robotics and Automation*, April (pp. 3478–3485). IEEE Press.
2. Batalin, M. A., & Sukhatme, G. S. (2003). Efcient exploration without localization. In *ICRA03: Proceedings of 2003 IEEE International Conference on Robotics and Automation*, May (pp. 2714–2719). IEEE Press.
3. Batalin, M., Sukhatme, G., & Hattig, M. (2004). Mobile robot navigation using a sensor network. In *ICRA04: Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, April (pp. 636–642). IEEE Press.
4. Batalin, M. A., & Sukhatme, G. S. (2003). Coverage, exploration and deployment by a mobile robot and communication network. In *Proceedings of the International Workshop on Information Processing in Sensor Networks*, April (pp. 376–391). ACM.

5. O'Hara, K. J., & Balch, T. R. (2004). Distributed path planning for robots in dynamic environments using a pervasive embedded network. In *AAMAS04: Proceedings of 2004 International Joint Conference on Autonomous Agents and Multiagent Systems*, August, ACM.
6. O'Hara, K. J., & Balch, T. R. (2004). Pervasive embedded networks for supporting multi-robot activities. In *Proceedings of the AAAI-04 Workshop on Sensor Networks*, July, AAAI Press.
7. O'Hara, K. J., Bigio, V., Dodson, E. R., Irani, A. J., Walker, D. B., & Balch, T. R. (2005). Physical path planning using the GNATs. In *ICRA05: Proceedings of 2005 IEEE International Conference on Robotics and Automation*, April (pp. 709–714). IEEE Press.
8. O'Hara, K. J., Bigio, V., Whitt, S., Walker, D., & Balch, T. R. (2006). Evaluation of a large scale pervasive embedded network for robot path planning. In *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May (pp. 2072–2077). IEEE Press.
9. Wagner, I. A., Altshuler, Y., Yanovski, V., & Bruckstein, A. M. (2008). Cooperative cleaners: A study in ant robotics. *International Journal of Robotics Research, 27*, 127–151.
10. Svennebring, J., & Koenig, S. (2004). Building terrain-covering ant robots: A feasibility study. *Autonomous Robots, 16*, 313–332.
11. Rekleitis, I., Dudeck, G., & Milios, E. (1997). Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *IJCAI97: Proceedings of the 1997 International Joint Conference on Artificial Intelligence*, (pp. 1340–1345), Morgan Kaufmann, August.
12. Icking, C., Kamphans, T., Klein, R., & Langetepe, E. (2005). Exploring simple grid polygons. In *COCOON05: Proceedings of the 2005 Annual International Conference of Computing and Combinatorics*, August (pp. 524–533). Springer: Berlin, Heidelberg.
13. Hazon, N., Mieli, F., & Kaminka, G. A. (2006). Towards robust on-line multi-robot coverage. In *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May (pp. 1710–1715). IEEE Press.
14. Ferranti, E., Trigoni, N., & Levene, M. (2007). Brick & Mortar: An on-line multi-agent exploration algorithm. In *ICRA07: Proceedings of the 2007 IEEE International Conference on Robotics and Automation*, April (pp. 761–767). IEEE Press.
15. Koenig, S., & Liu, Y. (2001). Terrain coverage with ant robots: A simulation study. In *AGENTS01: Proceedings of the 2001 ACM International Conference on Autonomous Agents*, May (pp. 600–607). ACM Press.
16. Choset, H. (2001). Coverage for robotics—a survey of recent results. *Annals of Mathematics and Artificial Intelligence, 31*, 113–126.
17. Kotz, D., Newport, C., Gray, R. S., Liu, J., Yuan, Y., & Elliott, C. (2004). Experimental evaluation of wireless simulation assumptions. Tech. Rep. Technical Report TR2004-507, Dartmouth College Computer Science, June.
18. Aguayo, D., Bricket, J., Biswas, S., Judd, G., & Morris, R. (2004). Link-level measurements from an 802.11b mesh network. In *SIGCOMM04: Proceedings of the 2004 ACM Conference of the Special Interest Group on Data Communication*, August (pp. 121–132). ACM Press.
19. Park, J., Park, S., Kim, D., Cho, P., & Cho, K. (2003). Experiments on radio interference between wireless LAN and other radio devices on a 2.4 GHz ISM band. In *VTC03: Proceedings of the 2003 IEEE Semiannual Vehicular Technology Conference*, April (pp. 1798–1801). IEEE Press.
20. Kotz, D., Newport, C., & E. C. (2003). The mistaken axioms of wireless-network research. Tech. Rep. Technical Report TR2003-467, Dartmouth College Computer Science, July.
21. Zhao, J., & Govindan, R. (2003). Understanding packet delivery performance in dense wireless sensor networks. In *SenSys03: Proceedings of the 2003 ACM Conference on Embedded Networked Sensor Systems*, November (pp. 1–13). ACM Press.
22. Cerpa, A., Busek, N., & Estrin, D. (2003). SCALE: A tool for simple connectivity assessment in lossy environments. Tech. Rep. Technical Report CENS-21, UCLA, September.
23. Ganesan, D., Krishnamachari, B., Woo, A., Culler, D., Estrin, D., & Wicker, S. (2002). Complex behavior at scale: An experimental study of low-power wireless sensor networks. Tech. Rep. Technical Report CSD-TR 02-0013, UCLA, February.
24. Hähnel, D., Burgard, W., Fox, D., Fishkin, K., & Philipose, M. (2004). Mapping and localization with RFID technology. In *ICRA04: Proceedings of 2004 IEEE International Conference on Robotics and Automation*, April (pp. 1015–1020). IEEE Press.
25. Kleiner, A., Prediger, J., & Nebel, B. (2006). RFID technology-based exploration and SLAM for search and rescue. In *IROS06: Proceedings of the 20062006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (pp. 4054–4059). IEEE Press, October.
26. Burgard, W., Moors, M., Stachniss, C., & Schneider, F. (2005). Coordinated multi-robot exploration. *IEEE Transactions on Robotics, 21*, 376–378.

27. Yanovski, V., Wagner, I. A., & Bruckstein, A. M. (2003). A distributed ant algorithm for efficiently patrolling a network. *Algorithmica, 37*, 165–186.
28. Wagner, I. A., & Bruckstein, A. M. (2001). From Ants to A(ge)nts: A special issue on Ant-robotics. *Annals of Mathematics and Artificial Intelligence, 31*, 1–5.
29. Wagner, I. A., Lindenbaum, M., & Bruckstein, A. M. (2000). MAC vs. PC—determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *International Journal of Robotics Research, 19*, 12–31.
30. Albers, S., Kursawe, K., & Schuierer, S. (1999). Exploring unknown environments with obstacles. In *Proceedings of the tenth ACM-SIAM Symposium on Discrete Algorithms*, January (pp. 842–843), ACM Press.
31. Batalin, M. A., & Sukhatme, G. S. (2007). The design and analysis of an efficient local algorithm for coverage and exploration. *IEEE Transactions on Robotics, 23*, 661–675.
32. Li, Q., De Rosa, M., & Rus, D. (2003). Distributed algorithms for guiding navigation across a sensor network. In *Mobicom 2003: Proceedings of 2003 ACM International Conference on Mobile Computing and Networking*, September (pp. 313–325). ACM.
33. Li, Q., & Rus, D. (2005). Navigation protocols in sensor networks. *ACM Transactions on Sensor Networks, 1*, 3–35.
34. Kong, C. S., Peng, N. A., & Rekleitis, I. (2006). Distributed coverage with multi-robot system. In *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May (pp. 2423–2429). IEEE Press.
35. Choset, H., & Pignon, P. (1997). Coverage path planning: The boustrophedon cellular decomposition. In *FSN97: Proceedings of 1997 International Conference on Field and Service Robotics*, December.
36. Yamauchi, B. (1998). Frontier-based exploration using multiple robots. In *AGENTS98: Proceedings of the 1998 ACM International Conference on Autonomous Agents*, May (pp. 47–53). ACM Press.
37. Howard, A., Parker, L. E., & Sukhatme, G. S. (2006). Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *The International Journal of Robotics Research, 25*, 431–447.
38. Burgard, W., Moors, M., Fox, D., Simmons, R., & Thrun, S. (2000). Collaborative multi-robot exploration. In *ICRA00: Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, April (pp. 476–481). IEEE Press.
39. Batalin, M. A., & Sukhatme, S. G. (2002). Spreading out: A local approach to multi-robot coverage. In *DARS02: Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems*, June (pp. 373–382).
40. Payton, D., Daily, M., Estowski, R., Howard, M., & Lee, C. (2001). Pheromone robotics. *Autonomous Robots, 11*, 319–324.
41. Zheng, X., Jain, S., Koenig, S., & Kempe, D., (2005). Multi-robot forest coverage. In *IROS05: Proceedings of the 2005 IEEE International Conference of Intelligent Robots and Systems*, (pp. 3852–3857). IEEE Press, August.
42. Agmon, N., Hazon, N., & Kaminka, G. A. (2006). Constructing spanning trees for efficient multi-robot coverage. In *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, May (pp. 1698–1703). IEEE Press.
43. Hazon, N., & Kaminka, G. A. (2005). Redundancy, efficiency, and robustness in multi-robot coverage. In *ICRA05: Proceedings of 2005 IEEE International Conference on Robotics and Automation*, April (pp. 735–741). IEEE Press.
44. Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: Part I, *Robotics and Automation Magazine IEEE, 13*, 99–110.
45. Bailey, T., & Durrant-Whyte, H. (2006). Simultaneous localization and mapping: Part II. *Robotics and Automation Magazine IEEE, 13*, 108–117.