Brick&Mortar: an on-line multi-agent exploration algorithm

Ettore Ferranti Niki Trigoni Mark Levene

***Abstract*— When an emergency occurs within a building it is critical to explore the area as fast as possible in order to find victims and identify hazards. We propose Brick&Mortar, an algorithm for the autonomous exploration of unknown terrains by a team of mobile agents. Because of the unreliability and short range of wireless communications in indoor environment we suggest that agents communicate indirectly with each other by tagging the environment. Agents have no prior knowledge of the map, but they are able to coordinate in order to explore a variety of terrains with different topological features. In our experimental evaluation, we show that Brick&Mortar significantly outperforms the competing algorithms, namely Ants and Multiple Depth First Search, in terms of exploration time. The observed performance benefits suggest that our algorithm is suitable for safety-critical applications that require rapid area coverage for real-time event detection and response.**

I. INTRODUCTION

Chemical, biological, radiological, nuclear and explosive (CBRNE) events refer to the uncontrolled release of chemicals, biological agents or radioactive contamination into the environment or explosions that cause widespread damage. CBRNE events can be caused by accidents or by terrorist acts. Although an event of this type could take place almost everywhere, the most challenging environment (and the one with the highest probability to occur) is on indoor environments, in a building or another highly frequented public place, such as a train station or the underground, as sadly demonstrated by the recent terrorist attacks in New York, Madrid and London.

After a CBRNE event, the area is off-limits and hazardous for everyone not wearing respiratory equipment, garments, barrier materials to protect themselves from exposure to biological, chemical, and radioactive hazards. This kind of suit could be very heavy and bulky, it could limit the respondents' movements, and reduce their sensing capacity (touch, vision and hearing).

Information gathering inside the area is essential to avoid risking the lives of the first responders: for example, if we knew the locations of the victims before entering a building, the responders could immediately get there avoiding hazardous areas such as rooms on fire or collapsed corridors or stairs. 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.

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. Such operations, however, may be obstructed by a number of limitations, e.g. the possible lack of a terrain map (the environment could anyway be heavily changed after 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 in the terrain, even if it were able to keep memory of its previous steps. In this paper, we take into account these limitations, and assume that agents can only rely on local information that is sensed in their vicinity (which other agents have left as trace), before making the next exploration step.

In this paper, we first analyze the functionality of two existing approaches to terrain exploration, namely Ants [1], [2] and Multiple Depth First Search (MDFS), and highlight their limitations. On the one hand, agents running the Ants algorithm cannot determine when the exploration task is completed. Moreover, whilst the first few agents are rapidly discovering new terrain, most of the remaining agents dwell on already explored network areas, leading to inefficient use of agent resources. On the other hand, agents running MDFS know when the exploration task terminates, but we show that they equally lack in coordination skills and often make poor use of resources.

Our novel algorithm, Brick&Mortar, 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 the three 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 II presents the assumptions of our model, and Section III provides a brief description of existing exploration algorithms and discusses their limitations. Section IV presents the new Brick&Mortar algorithm and discusses how it addresses the problem of agents traversing the same areas in loops. Section V presents a thorough experimental analysis of the three algorithms. An overview of related work is provided in Section VI, followed by conclusions and directions for future work in Section VII.

II. MODEL

In this section, we describe the model used by the proposed algorithm, Brick&Mortar, and the two competing approaches, Ants and Multiple Depth First Search, which we present in detail in Section III. We consider the task

E. Ferranti, N. Trigoni and M. Levene are with the School of Computer Science and Information Systems, Birkbeck College, University of London, UK { ettore | niki | M.Levene }@dcs.bbk.ac.uk

of exploring a hazardous terrain using a group of mobile nodes, hereafter referred to as *agents*. The overall area is divided into square cells, some of them representing walls. In our model, walls are used to identify both obstacles (e.g. victims), that agents cannot cross during their exploration phase, and real brick walls that constitute the building itself.

A cell can be in one of the following states:

- *Wall*: The cell cannot be traversed by an agent because it is a wall or it is blocked by an obstacle, so that the free space in it is not big enough to let an agent go past it.
- *Unexplored*: No agent has been in the cell yet.
- *Explored*: The cell has been traversed at least once, but the agents might need to go through it again in order to reach *unexplored* cells.
- *Visited*: The agents have already explored the cell, and they do not need to go through it again to reach other cells. Conceptually it is equivalent to a *wall* cell, in that no agent is allowed to traverse it.

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. In indoor environments where GPS cannot be used, agents do not rely on knowledge of their exact location; however, once they find themselves within a cell (covered by a stationary device), they can turn towards one of the four directions until they reach the next cell. Moreover, 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 state of the local cell.

Another characteristic of the model is that if agents are moved from their current location, they are able to resume their operations in the new area where they land. There is no centralized control of the agents' movements, so an exploration algorithm must be fully distributed. This means that agents make independent decisions about how to navigate through the terrain based on the local state. In fact we assume that the terrain is instrumented with uniformly distributed miniature devices (e.g. motes or RFIDs) capable of storing small amounts of information about the state of the local area. Certain areas that are occupied by walls or hazards will be void of such devices. In case some areas are not covered by any devices, agents can deposit such devices when they explore the area for the first time. In our initial work, we make the abstraction that the terrain is divided into square cells, and there exists at least one device per cell.

III. EXISTING ALGORITHMS

Before the presentation of our novel contribution - Brick&Mortar - we would like to give a brief description of two existing algorithms - Ants and Multiple Depth First Search - against which we test its performance in Section V. We also discuss some of their limitations that motivated us to design our new Brick&Mortar algorithm. To our knowledge, these are the only competing algorithms that do not rely on agents knowing their locations and the area map, and being able to establish reliable communication with each other. A detailed review of related techniques with slightly different assumptions is provided in Section VI.

A. Ants

We first discuss the behaviour, strengths and limitations of the Ants algorithm proposed in [1], [2]. This is a distributed algorithm that simulates a colony of ants leaving pheromone traces as they move in their environment [2]. 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).

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 realize it 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 exheeds a threshold, the counter is reset to 0.

The main limitation of the Ants algorithm is that the *visited* state is not used to mark the cells, so there is no way to tell when the environment is fully explored, 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. Another limitation concerns the inadequate use of agent capabilities: in a scenario with many rooms most of the agents are busy sweeping the first few rooms repeatedly while only a few of them set out to explore new areas.

A further drawback of the algorithm is the limited collaboration among agents. As is shown in Figure 1, in a scenario with many rooms most of the agents would sweep the first one or two 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.

B. Multiple Depth First Search

In order to address the limitations of the Ants algorithm, we consider a Depth First Search (DFS) approach to travers-

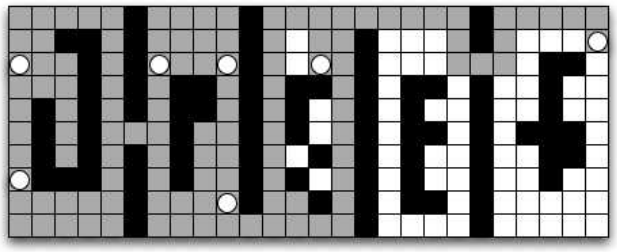

Fig. 1. The Ants algorithm is not efficient in a scenario with many rooms, because most of the agents explore the first rooms repeatedly, while only few of them set out to discover new areas.

ing the unknown terrain. 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 (e.g. 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, all cells are traversed exactly 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.

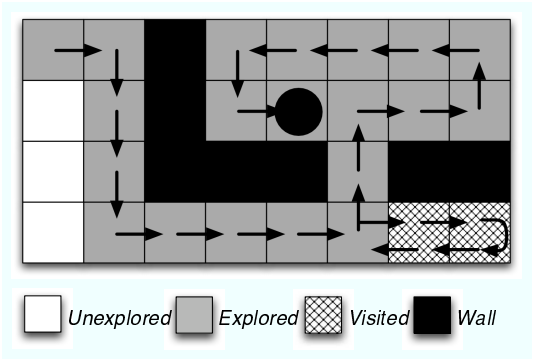

Fig. 2. Different branches in a Depth First Search exploration.

A snapshot of the exploration process with one agent can be seen in Figure 2: 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 realizes 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 (Algorithm 1), each agent builds its own exploration tree, and tries not to interfere with the trees of the other agents by marking *explored* cells with its own (agent) ID. When an agent finds itself at a cell surrounded by at least one *unexplored* cell or *explored* cell with the agent's ID, it can proceed as in the simple DFS algorithm. Otherwise, if one of the adjacent cells is annotated as *explored* by another collaborating agent, the current agent navigates through the *explored* cells of the collaborating agent's tree, trying to find an *unexplored* neighboring cell. If it finds one, it uses it as the root of a brand new tree that it starts covering with its own ID. Otherwise, if it becomes surrounded by *visited* or *wall* cells it terminates. MDFS requires $O(N)$ storage space at each cell, where N is the number of agents.

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. 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 (or exactly twice if we use only one agent), thus resulting in a long exploration time even in open areas without walls where a single traversal would suffice.

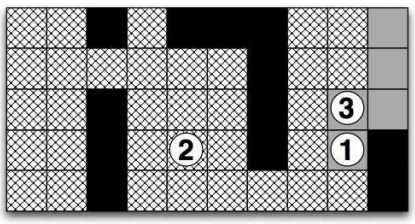

Fig. 3. Agents 1 and 2 are surrounded by *visited* cells and are therefore *trapped*, i.e. unable to assist agents 3 and 4 in the exploration task.

Another limitation of MDFS is that some of the agents

may become surrounded by *visited* cells before the whole area has been covered, resulting in a waste of the available resources. Notice in Figure 3 that at some point only a subset of the agents is able to continue the exploration process, whilst others (agents 1 and 2) remain idle. The reason is that each agent builds its exploration tree without considering how it interferes with the others, often resulting in isolating certain agents within *visited* areas (effectively acting as walls). Thus MDFS can no longer exploit the agents' capabilities once they are blocked. This limitation led us to seek for a new algorithm that leaves "corridors" of *explored* cells through which all agents would be able to reach any remaining unexplored areas in the network.

Algorithm 1 Multiple Depth First Search

```
1: if the current cell is unexplored then
2:     mark it as explored
3:     annotate the cell with your ID
4:     annotate the cell with the direction of the previous cell
       (parent cell)
5: end if
6: if there are unexplored cells around then
7:     go to one of them randomly
8: else
9:     if the current cell is marked with your ID then
10:        mark it as visited
11:        go to the parent cell
12:    else
13:        go to one of the explored cells randomly
14:    end if
15:
16: end if
```

IV. The Brick&Mortar Algorithm

Our novel algorithm, named Brick&Mortar, is designed to address the weaknesses of the existing algorithms. 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, they typically traverse each cell less than twice and they never get trapped within boundaries of *visited* cells.

The main idea behind Brick&Mortar is that of thickening the existing walls by progressively marking the cells that surround them as *visited*. Note that *visited* cells are equivalent to *wall* cells in that they can no longer be accessed. In the description of Brick&Mortar, we refer to *wall* and *visited* cells as inaccessible cells, and to *unexplored* or *explored* cells as accessible cells. Brick&Mortar 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. The main rule that an agent must obey locally is never to mark the current cell as *visited* if, by doing so, it blocks the path between two accessible cells.

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 described below. We are now in a position to introduce the details of the proposed Brick&Mortar algorithm with and without loop closure.

A. Brick&Mortar without loop closure

The states of a cell are exactly the same as the ones used in the Multiple Depth First Search Algorithm and described in Section II. Brick&Mortar consists of two discrete steps detailed in the pseudocode below (Algorithm 2). 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 giving priority to the *unexplored* cells around it.

In the marking step the agent updates the state of the current cell, choosing between the *explored* and *visited* states. The cell is marked as *visited* only if it is not blocking the way between two accessible (*explored* or *unexplored*) cells (say A and B) located in the North, East, South or West directions. In other words, the current cell is marked as *visited*, if there is an alternate path of accessible cells connecting A and B. Such alternate paths are easy to compute locally, because they are strictly confined to the 8-cell perimeter of the current cell. If such a path does not exist, the current cell is marked as *explored*, meaning that this or another agent can still move to it in the future.

In the navigation step the agent tries to move to an *unexplored* cell which is likely to be marked as *visited* in the next marking step, so that there will be no need for this or other agents to come back to it in the future. The best candidate is the *unexplored* cell with the greatest number of inaccessible (*visited* or *wall*) cells in its four directions. If there is no such candidate, the agent goes to one of the *explored* cells. If all four cells are inaccessible, it means that the terrain exploration has been completed.

Details of the two steps are provided in the pseudocode below:

B. Brick&Mortar with loop closure

The simple version of Brick&Mortar, described in the previous subsection, terminates successfully 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 clusters of *wall* cells in the middle of an area. For example, in Figure 4a, an agent on cell C_1 of the figure will start building a corridor of *explored* cells traversing cell C_2 and

Algorithm 2 Brick&Mortar Without Loop Closure

1: **Marking Step**
2: **if** the current cell is not blocking the path between any two *explored* or *unexplored* cells around **then**
3: mark the cell as *visited*
4: **else**
5: mark the cell as *explored*
6: **end if**
7: **Navigation Step**
8: **if** at least one of the four cells around is *unexplored* **then**
9: for each of the *unexplored* cells see how many *wall* or *visited* cells are around it, then go to the cell with most of them, which is most likely to be marked as *visited* in the marking step
10: **else if** at least one of the four cells around is *explored* **then**
11: go to one of them. Avoid selecting the cell where you came from unless it is the only candidate. Instead select the first *explored* cell in an ordered list of adjacent cells, e.g. [North,East,South,West] {The order of cells in the list depends on the agentID, so that different agents disperse in different directions}.
12: **else**
13: terminate {All adjacent cells are inaccessible, i.e. *visited* or *wall* cells}
14: **end if**

then finding itself back at cell C_1 again. According to the rule in the marking step of Algorithm 2 every cell blocks the path between the previous and the following one, and is thus repeatedly marked as *explored*. The loop problem is well known in the literature, but usually the proposed algorithms either ignore it [3] or need an external human operator to solve it [4]. In emergency scenarios, the team of agents might be inside a building or underground far away from the rescue team, and it should still be able to accomplish its exploration mission. For this reason we extend the original version of Brick&Mortar algorithm to enable loop closure without human intervention.

To make the algorithm capable of closing loops, we make an additional assumption: an agent is able to mark a cell with its ID (a simple number identifying the agent) and the directions (North, East, South or West) in which the agent is moving in and out of the cell. This assumption is not too strong because each agent can be equipped with a small and inexpensive electronic compass that detects and controls the direction of its movement. An agent detects the presence of a loop when it traverses the same *explored* cell twice in the same direction. If only one agent is performing the exploration task, it can easily break the loop by marking one of the cells as *visited* and then resuming its original wall-extension strategy (i.e. the marking and navigation steps described in Section IV-A).

However, if the terrain is explored by multiple agents, independent attempts to close the same or overlapping loops

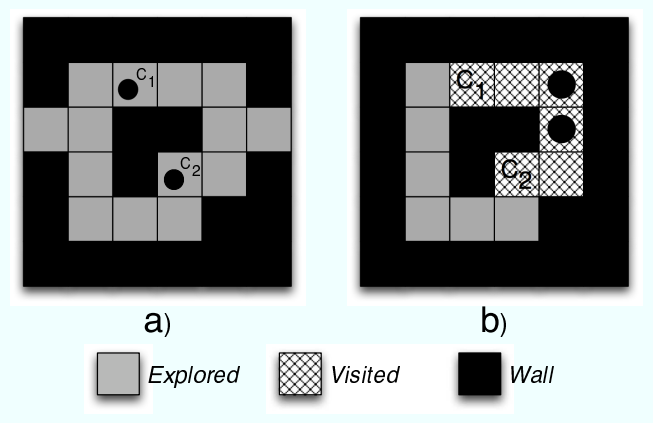

Fig. 4. The loop problem.

may result in agents being trapped within inaccessible areas and being unable to help with the remaining exploration task. For example, two agents A_1 and A_2 can traverse the loop of Figure 4a once starting from cells C_1 and C_2 respectively and moving in opposite directions. As they traverse the loop, they mark each cell as *explored*. Once agents A_1 and A_2 reach cells C_1 and C_2 respectively they detect the presence of a loop. To resolve the loop, they mark C_1 and C_2 as *visited*, and continue to move towards each other marking all cells in their way as *visited* (Figure 4b). Once they meet, they get trapped because they are surrounded by inaccessible (*visited* or *wall*) cells without having succeeded in marking the entire loop as *visited*.

- **Loop detection:** Initially, an agent follows the marking and navigation steps described in Section IV-A leaving a trace in each cell that it traverses (how it moved out of it), until it detects a loop. This happens when it moves into the same *explored* cell a second time (but not in the opposite direction than the one used previously to move out of that cell). Upon detecting a loop, the agent moves to the loop control phase. This phase requires $O(N)$ storage capacity at each cell, where N is the number of agents.
- **Loop control:** To take control of the loop, an agent A starts traversing the loop a second time in the same direction, trying to take control of each cell by annotating it with its own agent ID_A (this requires $logN$ bits, where N is the number of agents). It is possible only if the cell is explored and it is not already annotated with another agent's ID (say ID_B). Agent A knows that it has succeeded in taking control of all cells in the loop when it steps again on a cell that is already annotated with the same ID. In this case, it immediately switches to the loop closing phase.
- **Loop closing:** The agent is now in a position to break the loop by marking the current cell as *visited*, and continuing to do so until it reaches the first intersection (the cell with at least one *explored* neighbor cell that does not belong in the loop). The agent then switches to the loop cleaning phase.

- **Loop cleaning:** The agent removes any traces of the loop control phase, by moving backwards in the loop, removing its ID from the cells that it previously annotated, and resetting the direction in which the agent moved out of the cells to null.

It remains to describe in detail what happens in the loop control phase when an agent A is not able to take control of a cell, either because the cell is already *visited*, or because it is being controlled by another agent B. To deal with this scenario, we allow agents to interrupt the loop control phase, either permanently or temporarily. In the former case, an agent literally quits the loop resolution mechanism and moves to the loop cleaning phase. In the latter case, the agent can wait at a cell, as a *standby* agent, until the state of the cell is changed.

In what follows, we provide specific rules that define how an agent decides whether to continue, quit or stall controlling cells with its ID during the loop control phase. Let agent A try to move to the next cell in order to control it (i.e. to annotate it with ID_A). The decision it makes depends on the state in which it finds the cell:

$$\begin{cases} control\ cell, & if\ cell\ explored\ without\ control \\ start\ loop\ cleaning, & if\ cell\ visited \\ start\ loop\ cleaning, & if\ B\ controls\ \&\ (ID_B > ID_A) \\ start\ loop\ cleaning, & if\ C\ standby\ \&\ (ID_C > ID_A) \\ become\ standby, & otherwise \end{cases}$$

We finally need to define how, a standby agent (which waits at a cell as a result of following the last branch above), reacts to changes in the cells state. The rules that determine its behavior are:

$$\begin{cases} start\ loop\ cleaning, & if\ replaced\ by\ another\ agent \\ start\ loop\ cleaning, & if\ cell\ becomes\ visited \\ continue\ loop\ control, & if\ cell\ is\ cleaned \\ remain\ standby, & otherwise \end{cases}$$

An agent that manages to complete its loop control phase has succeeded in controlling *all common cells with other interfering loops*. It performs loop closing, and then cleans all common cells, before another interfering agent gets the opportunity to take control of them. Hence, although in general loops are handled concurrently, agents are able to detect when their loops interfere, and in this case, they resolve them in a sequential manner. The loop control phase has a similar role as locks in database systems, i.e. it allows concurrent operations whilst leaving the system in a *consistent* state. In our case, consistency means that all *explored* and *unexplored* cells remain connected, and no agent is trapped within *visited* cells.

Using the rules above we can prove that when agents try to resolve overlapping loops, they never cause a deadlock, they never get trapped within *visited* areas, and they always terminate (the proof is omitted for space reasons). However, to ensure this we require memory capacity at each cell that grows linearly in the number of agents (similar to MDFS storage requirements). In the future, we plan to investigate ways of adjusting our algorithm to require a fixed amount of storage space (like Ants).

V. SIMULATION RESULTS

We developed a simulation tool to test the performance of Brick&Mortar and the competing algorithms (Ants and MDFS), which allows us to automatically generate terrain maps with different topological features (by changing input terrain size, number of rooms, number of obstacles, etc.). The tool can be instructed to run any of the algorithms with different numbers of agents on a variety of maps.

In the results presented below, we study the impact of i) the number of agents, ii) the terrain size, iii) the number of rooms, and iv) the number of obstacles (that generate loops) on the performance of the three algorithms. Each point in the graphs is the average of running an algorithm 20 times, each time with a different 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 by 50) cells with 30 obstacles and 36 (6x6) 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* (black lines), i.e. the number of steps it takes for the agents to traverse all cells of the map at least once, which can be measured for all three algorithms; and ii) the *visiting time* (grey lines), i.e. number of steps that it takes for all agents to determine that they have completed the exploration task. The latter can only be measured for Brick&Mortar and MDFS, since agents runnings the Ants algorithm cannot determine when they terminate.

Effect of agents: Figure 5 shows that Ants always underutilizes agent resources compared to Brick&Mortar, but as the number of agents increases it becomes faster than MDFS. The MDFS algorithm is the least sensitive to varying the number of agents, which is owed to the fact that agents become trapped early on within visited cells and are unable to help cover new areas. Brick&Mortar uses efficiently up to 15 agents, beyond which point it does not improve much, owing to the fact that agents interfere with each other trying to resolve the same loops. Observe that the gap between the exploration and visiting time of Brick&Mortar increases with the number of agents. When more agents are available, they manage to speed up traversing the cells at least once, but they cannot do much to speed up loop closure and terminate early. In the future, we plan to study how we could achieve better load balancing by dispersing agents to enter the network from different cells.

Effect of area size: Figure 6 shows that Brick&Mortar scales gracefully as the area size increases. For areas of up to 4,900 cells, MDFS is 8 times slower and Ants 6 times slower than Brick&Mortar in terms of exploration time; similarly, MDFS is 4 times slower in terms of visiting time.

Effect of rooms: Figure 7 shows that as we increase the number of rooms the exploration time of the Ants and Brick&Mortar algorithms are not significantly affected, whereas the exploration time of MDFS becomes 2 times slower from 4 to 64 rooms. For an area of 40 rooms the exploration time of Brick&Mortar is half its visiting time,

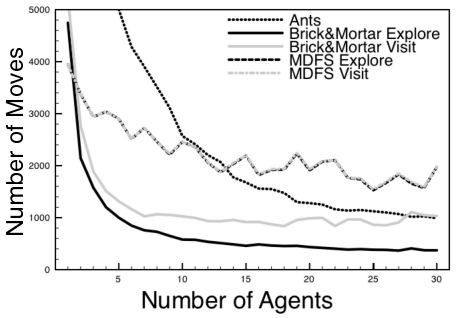

Fig. 5.

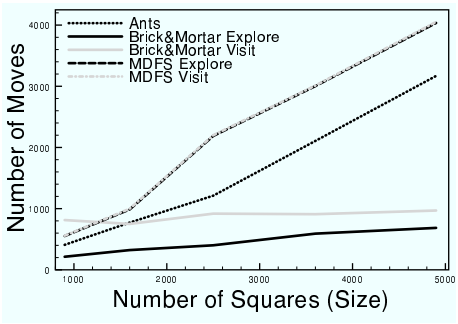

Fig. 6.

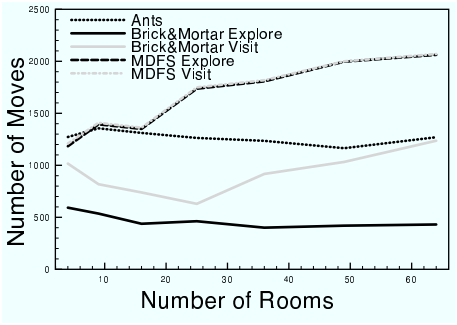

Fig. 7.

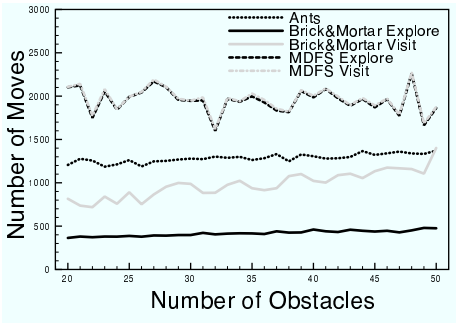

Fig. 8.

which means that the task is completed way before the agents know it. In all cases, Brick&Mortar outperforms the two competing approaches by factors of 3 or 4, both in terms of exploration and visiting time.

Effect of obstacles (wall cells that cause loops): Figure 8 demonstrates the impact of obstacles (and therefore loops) on the performance of the three algorithms. We observe that Ants and MDFS are not particularly affected by the presence of loops, whereas the visiting time of Brick&Mortar increases linearly in the number of obstacles. Hence in certain environments with a very large number of obstacles, it might be worth using one of the competing approaches, or a hybrid algorithm in which an agent starts with Brick&Mortar and switches to Ants or MDFS upon detecting a loop. We leave the study of hybrid algorithms to future work.

VI. RELATED WORK

Choset [5] 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. Zheng et al. [6] prove that the original problem is NP-complete, and propose a polynomial algorithm that yields a solution at most eight times slower than the optimal solution. Agmon et al. [7] propose a faster tree construction algorithm, while Hazon et al. [8], [9] 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. Our Brick&Mortar algortihm differs from these approaches in that it considers the on-line problem, i.e. it does not rely on the knowledge of the terrain's map. 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. 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. For example, the ants-inspired algorithm [1], [2] divides the area into square grid cells on which the exploring agents leave traces of their passage, similarly to real ants leaving pheromone. This algorithm is already described in detail in Section III. A similar approach to the Ants algorithm uses a sensor network infrastructure to provide agents with information about the visited areas and direct them to the least recently visited direction [10]. Kong et al. [11] propose an algorithm that explores an area in forward and reverse phases. The idea is to gradually build 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. Although this algorithm yields very efficient coverage time, it makes assumptions that are not always realistic in harsh environments, namely that agents have perfect wireless communication and they always know their position in the map. Yamauchi presents a frontier-based exploration algorithm [12], 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. The algorithm also makes the same strong assumptions as in [11], namely perfect localization and reliable communication among agents. Burgard et al. do not assume that the environment is divided into grid cells [13]. Agents compute a utility function to go to the next "frontier" in order to maximize the explored territory. The agents have a list of target points to reach in the next step, each of them associated to a value which takes in 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 in 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. [14] 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. Batalin et al. [15] focus on agent dispersion and propose two algorithms to make the agents move away from each other when they are in sensing range. Finally, Howard et al.[4] present a general approach to exploring a building, finding objectives and reporting 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.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a new algorithm called Brick&Mortar, for the multi-agent exploration of unknown terrains. Agents running our algorithm can easily determine when the exploration task is completed. Our algorithm avoids exploring the same areas multiple times, it makes good utilization of all agents and it is capable of resolving loops. The experiments show that our algorithm is significantly faster than the two competing algorithms, Ants and Multiple Depth First Search, in a variety of scenarios.

In the future we would like to adjust the exploration algorithm, in order to maintain communication channels between the exploring team of agents and the human personnel outside the building. Once an agent detects a hazard or a survivor, it must immediately report this event to the rescue team, before continuing its exploration. Another challenge is to cope with unexpected events like collapsing walls that block some of the cells, or relocation of agents to different parts of the terrain (e.g. by the rescue team). We would also like to relax the requirement of having at least one miniature device per cell, and consider algorithms that enable fast exploration of sparsely-instrumented environments. Finally, mobile nodes could leverage their limited storage capabilities to maintain partial views of the terrain's map in their memory. Once they are within communication range, they could collaborate by carefully merging their inaccurate and possibly inconsistent maps.

VIII. ACKNOWLEDGMENTS

Thanks to Elisa Rondini who suggested an optimization of the Brick&Mortar loop closure phase, to Sven Helmer for useful comments, and to Andrea Ciresa (Italian Red Cross) for the advices concerning his experiences on real emergency scenarios.

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 Governmental purpose notwithstanding any copyright notation thereon.

REFERENCES

[1] S. Koenig and Y. Liu, "Terrain coverage with ant robots: a simulation study," in *AGENTS01: Proceedings of the fifth international conference on Autonomous agents*. ACM Press, 2001, pp. 600–607.

[2] J. Svennebring and S. Koenig, "Building terrain-covering ant robots: A feasibility study." *Auton. Robots*, vol. 16, no. 3, pp. 313–332, 2004.

[3] C. Icking, T. Kamphans, R. Klein, and E. Langetepe, "Exploring simple grid polygons," in *COCOON 2005: Proceedings of Computing and Combinatorics: 11th Annual International Conference*. Springer Berlin / Heidelberg, 2005.

[4] A. Howard, L. E. Parker, and G. S. Sukhatme, "Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection," *The International Journal of Robotics Research*, vol. 25, pp. 431–447, 2006.

[5] H. Choset, "Coverage for robotics - a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113–126, 2001.

[6] X. Zheng, S. Jain, S. Koenig, and D. Kempe, "Multi-robot forest coverage," in *IROS05: Proceedings of the International Conference of Intelligent Robots and Systems*. IEEE press, 2005.

[7] N. Agmon, N. Hazon, and G. A. Kaminka, "Constructing spanning trees for efficient multi-robot coverage," in *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation*. IEEE press, 2006.

[8] N. Hazon and G. A. Kaminka, "Redundancy, efficiency, and robustness in multi-robot coverage," in *ICRA05: Proceedings of IEEE International Conference on Robotics and Automation*. IEEE press, 2005.

[9] N. Hazon, F. Mieli, and G. A. Kaminka, "Towards robust on-line multi-robot coverage," in *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation*. IEEE press, 2006.

[10] M. A. Batalin and G. S. Sukhatme, "The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment," in *ICRA05: Proceedings of IEEE International Conference on Robotics and Automation*. IEEE press, 2005.

[11] C. S. Kong, N. A. Peng, and I. Rekleitis, "Distributed coverage with multi-robot system," in *ICRA06: Proceedings of the 2006 IEEE International Conference on Robotics and Automation*. IEEE press, 2006.

[12] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Agents98: Proceedings of the Second International Conference on Autonomous Agents*. ACM press, 1998.

[13] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *ICRA00: Proceedings of the 2000 IEEE International Conference on Robotics and Automation*. IEEE press, 2000.

[14] I. Rekleitis, G. Dudeck, and E. Milios, "Multi-robot exploration of an unknown environment, efficiently reducing the odometry error," in *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1997.

[15] M. A. Batalin and S. G. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *DARS02: Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems*, 2002.