Multi-robot exploration of unknown environments with identification of exploration completion and post-exploration rendez-vous using ant algorithms

Mihai Andries, François Charpillet

Multi-robot exploration of unknown environments with identification of exploration completion and post-exploration rendez-vous using ant algorithms

Mihai Andries[1,2,3] and François Charpillet[1,2,3]

***Abstract*— This paper presents a new ant algorithm for the navigation of several robots, whose objective is to autonomously explore an unknown environment. When the coverage is completed, all robots move to a previously defined meeting point. The approach that we propose in this paper for solving this problem, considers that the robots build, while moving, a common and shared representation of the environment. In this representation, the environment is viewed as a graph (typically a set of connected cells in a regular grid), each grid cell having a local memory able to store a limited amount of data. A robot can write numbers on the cell on which it is lying. It can also read the values of the cells in its neighborhood, and perform some simple operations, such as computing the minimum of a set of values. Each robot is capable, contrary to most ant-based approaches, to determine, in a distributed way, when the environment coverage has completed. Few ant algorithms can do that. Brick&Mortar is one of them and this is why it retains a central place in our proposition. The novelty of our approach is that, due to an emerging property of the underlying algorithm, agents will finish their exploration at a pre-defined evacuation point. In addition, several improvements of the original Brick&Mortar algorithm are proposed in this paper, such as the possibility to use better local strategies at the robot level (using, for example, LRTA*). The paper also presents a set of benchmarks against the best existing ant algorithms on several widespread graph topologies.**

I. Introduction

Graph exploration is a topic with numerous applications, such as robotic exploration of buildings, robotic navigation for home automation and web indexing spiders. In the context of robotic exploration of buildings, three problems have to be solved: (1) guaranteeing full exploration of the environment (that can be represented as a graph), (2) identifying if a full exploration of the environment has been achieved, and (3) returning to the rendez-vous point (e.g. entry/exit point or evacuation point). Step (2) is critical for deciding when to execute step (3). Unfortunately, this is not accounted by most ant algorithms proposed in the literature, to which Brick&Mortar (see [3]) is an exception. Thus, this paper presents a new ant algorithm, Brick&Mortar Improved, for distributed exploration of graphs with no prior knowledge of the graph. Ant algorithms are multi-robot algorithms, that use pheromone traces for communication. Initially based on Brick&Mortar, an algorithm capable of identifying when environment exploration has ended, the proposed approach possesses a new feature of setting a post-exploration rendez-vous point. It also improves several aspects of Brick&Mortar, sensibly reducing the number of steps till the identification of exploration completion.

[1]Inria, Villers-les-Nancy, F-54600, France
[2]Université de Lorraine, LORIA, UMR 7503, Vandoeuvre-les-Nancy, F-54506, France
[3]CNRS, LORIA, UMR 7503, Vandoeuvre-les-Nancy, F-54506, France
firstname.lastname at inria.fr

II. Related work

There are currently three main families of algorithms for multi-agent exploration: (1) tree-cover algorithms, (2) frontier propagation algorithms and (3) ant algorithms.

Tree-cover algorithms use a pre-calculated spanning-tree to direct the exploration effort and distribute it among the agents. These algorithms require *a priori* knowledge of the environment. A typical example is the Multi-Robot Forest Coverage (MFC) algorithm, described in [21].

Frontier propagation algorithms use value propagation starting at the frontiers, to calculate how the exploration effort will be distributed among the agents. Some examples are described in [1] and [9].

Ant algorithms are different from the previous two categories in that they are distributed, multi-agent and robust by definition. They can adapt to an unknown environment, and they scale well with an increasing number of agents. Agents have no complex computations to make and they don't have to know their position. They use no inter-agent communication except marks left on the ground. The implementations mentioned in the literature make use of thermal trails (see [13] and [2]), alcohol trails (see [15]), odour trails (see [14], [12] and [11]) and ink markings (see [18]). Another proposed implementation makes use of stationary sensor nodes, which are dropped by the agents and used to relay information about the environment (see [4]). Nevertheless, the use of a terrain-marking system is also the biggest weakness of the ant-based approach, in that the practical amount of data that can be stored in this way is limited. A compromise solution to this problem would be to loosen the *no communication hypothesis* and use a shared memory (a method also called *virtual trails* in the literature). This approach allows the storage of more information than the traditional marking methods would have permitted. On the other hand, this opens the door to potentially more powerful approaches. Nevertheless, trace-leaving algorithms can be applied in digital environments, such as web-crawling graphs (see [16]), or on intelligent tiles used in house automation, that can store data in their own memory (see [10]), as well as on interactive smart surfaces (see [17]).

Ant algorithms exist under several forms, the two dominant ones being *vertex marking algorithms* and *edge marking algorithms*. In the traditional representation, generic marking ant algorithms can be resumed by the following scheme: (1) initialize all the marking spots, (2) choose a starting vertex, (3) choose an edge to follow, (4) leave a trace on the marking spot using the value-update rule, (5) traverse the chosen edge to arrive to the new vertex, (6) go to 3. The marking spots can be defined as either vertices (for vertex marking algorithms) or edges (for edge marking algorithms).

III. Phases of exploration

Distributed exploration of an unknown environment comprises several checkpoints: (1) start of the exploration, (2) achievement of full exploration of the environment, (3) identification of this achievement and (4) return to an evacuation point. The time it takes an agent, following a given algorithm, to pass throught these checkpoints can be aproximated using theoretical time-bounds.

Cover time is the time it takes for a team of robots to complete the environment exploration. It is upper-bounded by the time it takes a single robot to reach a target-point of the environment, in the worst case (see [8]). The cover time is influenced by the number of robots participating in the exploration, the size of the environment, its topology (open spaces or corridors) and by the starting point of the exploration.

Termination awareness is the capacity to identify, in a distributed way, when the exploration has been completed. This indicator is important for agents exploring an environment, as opposed to those performing continuous patrolling.

When measuring the efficiency of a multi-agent exploration algorithm, usually two possible initial deployments of agent teams are considered: the case of a clustered (grouped) team, and the case of a scattered team. Scattered teams are those that are initially (equally) distributed throughout the environment. Clusterd (grouped) teams start their exploration from a single point, like a team that enters a building through the main entrance. Examples of both approaches can be found in [5].

Although the scattered deployment allows to make several considerable simplifications of the model, for calculating the covering-time bounds, it is a quite unrealistic approach. To fit this model in practice, for example, robots would have to be parachuted over the environment. This is applicable for urban rescue situations, but is not suited for the exploration of abandoned mines.

On the other hand, the case of grouped teams that start their exploration of a building by entering through a door seems more plausible in practice.

Therefore, only the case where agents are initially grouped before starting their exploration will be analyzed in this article.

IV. Brick&Mortar

Brick&Mortar is an ant algorithm introduced in [3], that is currently the only one (to our knowledge) capable of identifying the completion of the exploration process. This algorithm perceives the environment as a grid, in which the agents move, using 4-connectivity rules. In its representation of the environment, Brick&Mortar uses 4 types of cells: *unexplored* nodes, *explored* nodes (visited nodes which connect the unexplored regions), *closed* nodes (visited nodes that will be avoided during further exploration of the environment)[1] and *walls* (impenetrable nodes).

During the exploration phase, directing towards *unexplored cells* is prefered, compared to exploring other types of cells. When surrounded only by *explored cells*, an agent will choose the one which was visited the least number of times. This heuristic for dispersing agents is known as *Node Counting* in the literature. Access to *closed cells* is forbidden.

Brick&Mortar uses a *tabu list* approach (see [7]) to mark as *closed* the cells of the grid, that are not needed for further exploration of the environment. When all the cells of the grid have been marked as *closed*, the exploration of the environment is considered complete.

A cell can only be marked as *closed* when, in the viewing range of the agent, it does not cut the graph into pieces if removed (when it does not form a bridge between two connected components of the graph). This guarantees that all the *non-closed* regions are strongly connected (that the graph is never cut into separated, disconnected pieces). This has the emergent effect of keeping all the exploring agents inside the graph that contains all the *non-closed* regions. Thus, as the size of the *non-closed* graph reduces to zero, all the agents will concentrate in a single point (given the assumption that N robots can physically share a cell). Therefore, agents will be able to detect when the exploration of the environment has finished, by checking if all the surrounding cells are marked as *closed*.

The algorithm is capable of taking into account an additional parameter: the viewing range of the agent. This implies that when attempting to *close* a cell, agents check the connectivity of the graph only inside their viewing range.

However, when a robot finds itself inside a closed circuit, none of the cells forming the circuit can be closed without violating the *graph-connectivity* condition. This creates a loophole in the algorithm, generating a situation where cells cannot be closed. This kind of circuits form around isolated obstacles in the environment. Brick&Mortar uses a distributed *loop closing* algorithm for solving these situations. This problem is specific to agents with a limited viewing range.

Upon detecting its own trace on an already visited cell, an agent will launch its *loop closing* algorithm, in order to check if the loop is still present, lock the loop for its own use, mark the loop as closed, and remove the lock.

The locking of cells requires managing the priority between the agents. For this scope, Brick&Mortar uses the IDs of agents in order to decide which agent will continue its *loop closing* algorithm, and which one will have to wait.

[1] In the original paper by [3], *closed* nodes are called *visited* nodes. Their naming was changed to avoid confusion.

When an agent succeeds in locking the loop that it detected, it enters the *loop closing* phase. In order to respect the graph connectivity requirement, an agent will interrupt its *loop closing* phase when it crosses a cell that, inside agent's viewing range, forms a bridge, that links the loop to the rest of the graph. Whether the *loop closing* phase has been interrupted or successfully finished, the agent enters the *loop cleaning* phase, in which it removes its locks from the cells of the identified loop.

V. Brick&Mortar Improved (BMI)

Brick&Mortar Improved (BMI) is a new algorithm that we propose, built upon the Brick&Mortar algorithm. It introduces a new feature for gathering the agents at a rendez-vous point, once the exploration is complete. In addition, it brings some notable improvements in terms of execution time to classic Brick&Mortar. It does so by optimizing its *loop closing* algorithm, that deals with the problems posed by an agent's limited viewing range. It introduces identification of false positive detections of loops. It also modifies the heuristic employed for dispersing the agents throughout the environment. Section V-A deals with the rendez-vous task, while this section concentrates on the optimizations brought to the *loop closing* algorithm.

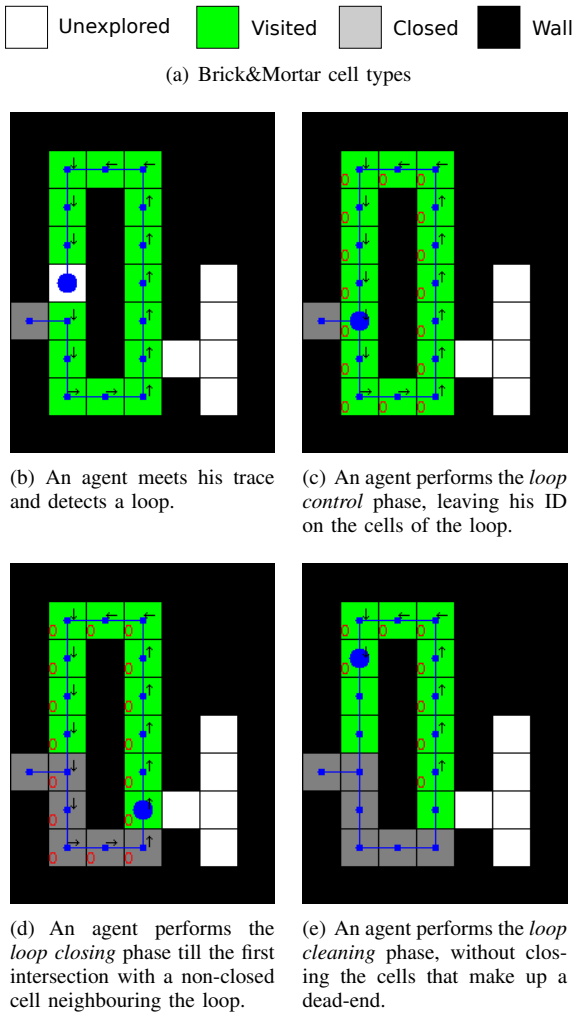
Firstly, during the *loop closing* phase, agents stop closing the cells of the loop at the first intersection with a non-closed cell, not belonging to the identified loop. This allows to maintain the (strong) connectivity of the overall graph, without risking to separate the loop from the rest of the graph. However, if an agent starts closing the loop at such an intersection, the *loop closing* phase will directly interrupt, without having closed any cell in the identified loop.

In the case of maps filled with obstacles arranged in a grid (see, for example, figure 7(a)), this poses a serious problem, as the exploration time may dramatically increase due to the frequent inefficient use of this heuristic. This is partially due to the behaviour that prioritises *loop closing* over exploration. We thus propose to improve this behaviour, when agents quit prematurely their *loop closing* algorithm.

The proposed solution is to continue the *loop closing* phase if it has started at an intersection, by skipping the intersection without closing it, and by interrupting eventually only at the second intersection (if such occurs). This solution guarantees that after each *loop closing* phase, at least one cell will be closed (at least one cell shall be added to the Tabu list).

Secondly, agents do not close cells behind them as they come out of a dead-end, during the *loop cleaning* phase (see figure 1 for an example scenario). This can be viewed as a leak in the efficiency of the algorithm. The proposed solution is simple, and implies checking whether, by closing a cell during the *loop cleaning* phase, the remaining graph stays connected inside the agent's viewing range. However, as in the case of *loop closing*, this should be done only until the first link with the rest of the graph, while moving in reverse sense, in order to maintain the graph connectivity.

Finally, agents can detect false positives (inexistent loops), when they encounter an old trail of their own. We have



(a) Brick&Mortar cell types

(b) An agent meets his trace and detects a loop.

(c) An agent performs the *loop control* phase, leaving his ID on the cells of the loop.

(d) An agent performs the *loop closing* phase till the first intersection with a non-closed cell neighbouring the loop.

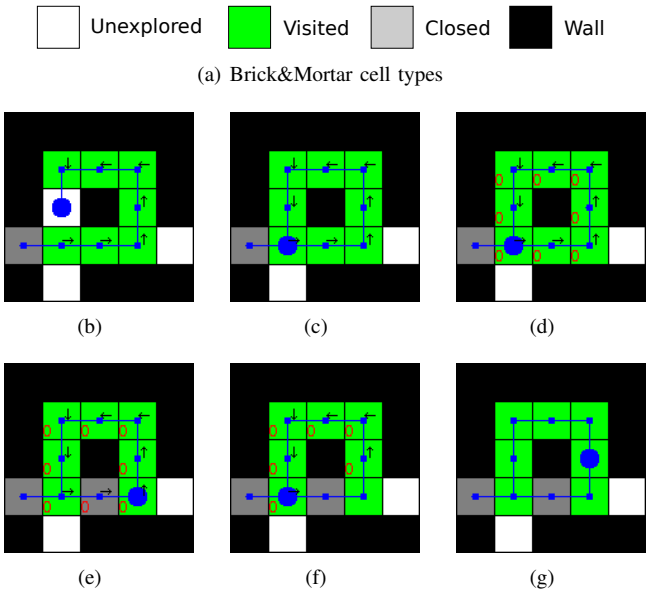
(e) An agent performs the *loop cleaning* phase, without closing the cells that make up a dead-end.

Fig. 1. An example of a dead-end left open by an agent after the *loop cleaning* phase.

identified three types of loops, that agents can recognize: (1) a (true positive) loop, detected without having previously closed any cells between two visits of a cell in the loop; (2) a (false positive) loop, detected after having closed one or several cells in the *loop detection* mode (in other words, cells that weren't part of a loop); (3) a (complex true positive) loop detected having closed one or several cells in the *loop closing* phase.

Loops of the first type are normally resolved by the classic Brick&Mortar algorithm. Each of these loops contains a path, from the vertex on which the loop was detected, and back to this same vertex, with a length equal to the number of steps that the agent has made since it last visited it. This route is only available if the agent has not performed any cell-closing activity since the last visit of this vertex. Otherwise, the loop would have been cut by these closed cells.

The second type of detectable loops, called *cut loops* (see figure 3), causes the Brick&Mortar algorithm to identify a false positive presence of a loop, which is in fact *cut* by the cells that were closed between the visits of the cell, that served as entrance into the loop. The agent would have



(a) Brick&Mortar cell types

(b) (c) (d)

(e) (f) (g)

Fig. 2. Solution for a premature exit from the *loop closing* phase. 2(b) An agent on the verge of detecting the presence of a loop. 2(c) An agent detects the presence of a loop. 2(d) An agent ends the *loop control* phase, leaving an ID trace on the cells over which it has taken control. 2(e) An agent executes the *loop closing* phase. It skips the first intersection, because the *loop closing* phase has started at an intersection, and doesn't stop until the second intersection. 2(f) An agent performing *loop cleaning*, going backwards through the loop. 2(g) An agent ends its *loop cleaning* phase (and quits the loop resolution algorithm), by returning to the "normal mode" of *loop detection*. The classic Brick&Mortar algorithm would have stopped its *loop closing* phase at step 2(d), ending its loop resolution algorithm without having closed any of the cells in the identified loop.

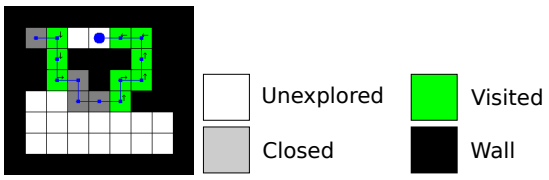
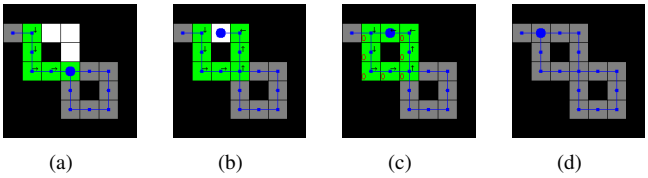


Fig. 3. Example of detection of a *cut loop*: an agent encounters its own trace and detects a false positive loop.

consequently lost time by engaging into the costly loop resolution algorithm, which is guaranteed to stop prematurely under these conditions, without having closed any cells in the end. This implies that an agent can recognize *cut loops*, if the timestamp, that the agent left on the last cell it has closed, is more recent than the one found on the re-encountered cell.

The third type of detectable loops, called *reduced loops* (see figure 4) contains a portion of a secondary loop that was encountered and closed during the passage of the primary loop. Such a loop is not an issue for the basic Brick&Mortar algorithm, as it doesn't cut the circuit of the loop but only reduces its length. However, this type of loops should be distinguished from *cut loops*, as they should be engaged directly once encountered, compared to the *cut loops* where loop control and closing should be avoided as being useless.

These improvements do not break the general structure of



(a) (b) (c) (d)

Fig. 4. Example of a reduced loop composed of two smaller interwiven loops. 4(a) An agent identifies and closes a first loop. 4(b) An agent identifies the second loop. 4(c) An agent takes control over the second loop. 4(d) The agent closes the second loop, cleans his traces and halts.

the algorithm (which was mentioned as proven to achieve complete exploration in [3]), because the state-machine used by the agents is left unchanged. Although we provide no formal proof of this statement, no deadlock was observed throughout the benchmarking phase, which consisted of over 500 runs of the algorithm.

A. Definition of a post-exploration rendez-vous point

A useful property for a multi-agent ant algorithm is to be able to set a rendez-vous point on the map, where the robots should return after the completion of their exploration objective. This, for instance, can allow a team of robots exploring a building to return to the entrance, once the exploration is complete. Such a behaviour can be easily coded, so that the returning phase emerges as a result of the employed navigation and marking rules.

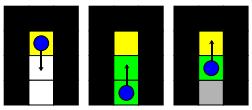
In the case of the Brick&Mortar algorithm, agents become aware of the completion of their exploration objective and remain motionless on their positions, because they possess no return path planification algorithm.

The idea here is to create a new type of cell marking, called an *attractor* cell, that can be closed only when surrounded by either closed cells or walls. In the context of Brick&Mortar, this implies that the *attractor* will be the last cell to be closed. Additional navigation rules are specified, that define the attractor as an explored cell, as well as additional marking rules, that specify that the attractor cannot be *closed*, if it isn't surrounded exclusively by closed cells or walls (in other words, before the exploration is finished). Thus, agents will have to leave a path towards this attractor, and will have to return to it in order to complete the exploration objective.

On the implementation level, two things change in the markings employed by BMI compared to Brick&Mortar: (1) the attractor is the last cell to be closed and (2) the attractor cell is treated as an intersection during the *loop closing* phase, when occuring inside the loop resolution algorithm. A comparative example of execution of Brick&Mortar and BMI is presented in figure 6. The performances of BMI are analysed in more detail in section VI-C.

VI. Experimental results of simulations in 2D environments

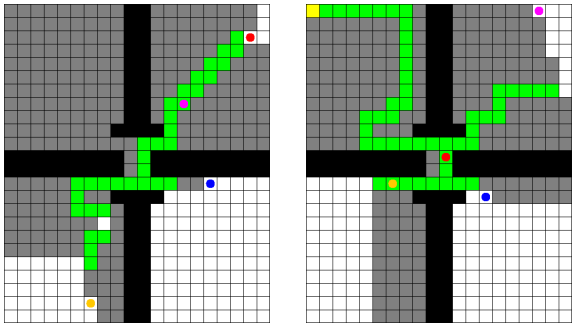
For our benchmarking results, a series of assumptions were made: (1) agents have four directions of movement (N, S, E, W); (2) agents' viewing range is 1 cell (in a 2D environment,



(a) (b) (c)

Fig. 5. Additional marking rules used by Brick&Mortar Improved (BMI). The attractor is treated as a non-explored region. When leaving the attractor, leave a path of explored cells back to it, as if the attractor was a non-explored region.

(a) Brick&Mortar Improved cell types.

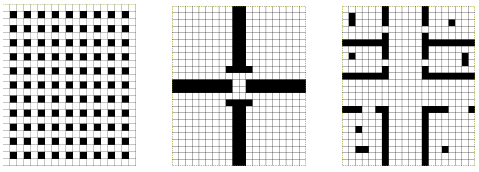


(b) (c)

Fig. 6. Comparative example of execution: Brick&Mortar vs BMI. 6(b) Execution of Brick&Mortar on the *4 rooms* map. It isn't possible to predict where will the agents be located after the exploration completion. 6(c) Execution of BMI on the *4 rooms map*. Agents leave a return path to the attractor cell.

they can see the cell underneath and the 8 neighbouring ones); (3) one cell can be shared among several agents (this was done to be able to compare our results with the ones previously presented in the literature); (4) the time consumed for leaving traces is negligible (otherwise we would have ended up counting the number of algorithm calls, instead of counting the total distance travelled by the agents); (5) the time complexity is given in the number of time-steps necessary for the agents to complete the exploration.

We have chosen 3 different types of maps, designed to evidentiate the weaknesses of algorithms: a map with obstacles of small size (figure 7(a)), a map with 4 rooms and no obstacles, each having a single entrance/exit (figure 7(b)), and an office building map (figure 7(c)). For each map type, we ran the algorithms on maps of 3 different sizes: small (1X), medium (2X), large (4X).



(a) (b) (c)

Fig. 7. Map types used for performance tests: 7(a) *Obstacles* map, 7(b) *4 rooms* and 7(c) *Office* map.

A typical robotic reconnaissance mission can be decomposed into 3 stages: (1) exploration of the environment, (2) identification of exploration completion and (3) gathering the agents after the end of exploration at a given rendez-vous point. We shall analyse these stages to design an algorithm capable of solving these 3 sub-problems, and at the same time performant enough to compete with the other algorithms, that solve only the exploration problem.

Therefore, we have structured our benchmarking process in 3 steps: (1) benchmarking of pure exploration algorithms, (simplyfying more complex algorithms like Brick&Mortar to place all algorithms in the same conditions), (2) benchmarking of algorithms capable of identifying exploration completion, (3) benchmarking of the new algorithm capable of identifying the exploration completion and returning to a rendez-vous point.

A. *Environment exploration*

Most algorithms proposed in the literature are only capable of performing a continuous exploration (patrolling) of the environment. The problem of covering an environment is close to the patrolling problem, the difference being that a single exploration of the environment is made.

An external observer is usually used to measure the performance of patrolling algorithms, applied to the problem of covering an environment. This is because most algorithms are not capable of identifying the termination. The only exception (to our knowledge) for the multi-agent case is constituted by the Brick&Mortar algorithm. The algorithm Ant-Walk-2 is also capable of identifying the termination of exploration, but only in the single agent case.

Given that the capacity of identifying exploration termination comes at an additional cost in terms of performance, it was decided to rewrite Brick&Mortar in such a way, so as to remove the termination identification part, leaving only the navigation and marking parts.

A detailed analysis of the Brick&Mortar algorithm has allowed us to separate it into 2 pieces: (1) the heuristic employed for dispersing the agents: choose the non-explored cell with the largest number of surrounding walls or closed cells; apply a gradient descent method like *Node Counting* (see [19]) to navigate among explored cells, (2) the loop resolution algorithm and the marking used for identifying the termination of exploration (the usage of a Tabu list). We were therefore able to create a simplified version of Brick&Mortar, that we have called *Brick&Mortar Simplified (BMS)*, using only the first part of the original algorithm. This has allowed us to see if the heuristic employed by Brick&Mortar was more efficient, compared to other algorithms.

First, we have compared the forementioned algorithms (see figures 8(a) and 8(b)), at which stage it was clear that Brick&Mortar was among the most performant algorithms. It surpassed all the other algorithms on all maps, except for the *Obstacles* map, which was specifically designed as a hard case for Brick&Mortar. We noticed that, on maps without obstacles (figure 8(a)), BMI behaves better that all other benchmarked algorithms. The explication hides in the

heuristic employed by Brick&Mortar, for navigating among non-explored cells (i.e. choose the one with most closed cells or walls around), that allowed it to travel through the environment by sticking to its perimeter walls, and thus generate a quasi-optimal path.

The weak performance of Brick&Mortar on the *Obstacles* map is due to the prioritisation of loop resolution over exploration. We tried to solve this problem, which appears on maps with lots of obstacles, by modifying the loop resolution algorithm described in section V.

We also found that we could augment the performances of Brick&Mortar by changing its heuristic for dispersing the agents on the map. We tried replacing the underlying Node Counting algorithm with other algorithms of the same family (e.g. Learning Real-Time A* (LRTA*), EVAW, Vertex-Ant-Walk with 2 traces, and the Thrun rule) and performed experiments to identify the fastest explorer. The sole interesting result is presented in figure 8(c), showing a very slight increase in performance when adapting LRTA* as the algorithm for agent dispersion. At the same time, using timestamps for dispersing agents proved counterproductive: when varying the number of agents on a *Obstacles* map of fixed size (20x20), it took between 1.5x and 2x more steps to attain the exploration objective using this dispersion algorithm, compared to all other variations of BMS. Although the advantage of the LRTA* dispersion heuristic is light, compared to all the other competitors, we chose to implement it (instead of Node Counting) in the new BMI algorithm.

B. Identification of exploration completion

Brick&Mortar and BMI were chosen for the performance tests in this category, as they are the only known algorithms capable of identifying the exploration completion. To avoid the overprice brought by the additional functionality of gathering agents at a rendez-vous point, after the end of the exploration, we have used a simplified version of BMI, called BMIS (which stands for BMI Simplified). BMIS is an optimised version of Brick&Mortar, but which does not perform the rendez-vous task.

Brick&Mortar and BMIS have the same performance on maps with no obstacles, a result that naturally confirms the theoretical expectations (as the optimised *loop closing* algorithm is never called). The results are the same, because the improvements of BMIS targeted the loop resolution algorithm, that is not emloyed on maps without obstacles.

On the other hand, the situation changes in environments heavily filled with obstacles, where the gain in time before the identification of exploration termination is clearly visible (see figure 8(d)). On an map of type *Office*, of size 20x20, when varying the number of agents participating in the exploration, BMIS needs, on average, between 30% and 13% less steps to identify the termination of exploration, compared to classic Brick&Mortar. The gain in exploration time of BMIS on the same map reaches 5-10% for 1 to 9 agents, after which the performances of the two algorithms equalize.

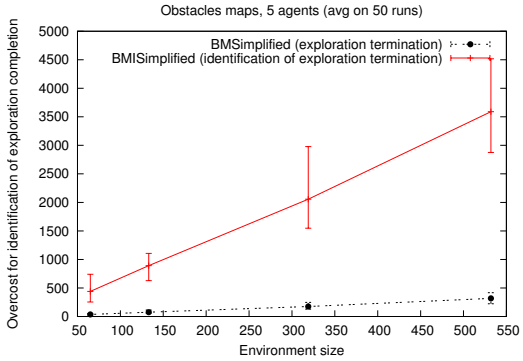


Fig. 9. This figure presents the overcost (in terms of time) brought by the identification of exploration completion, compared to a simple exploration of the environment. It is calculated by substracting the time taken by BMS to simply explore the environment from the time taken by BMIS to identify the exploration completion. The vertical lines show the min and max time the algorithms took to complete the objective, registered over 50 runs.

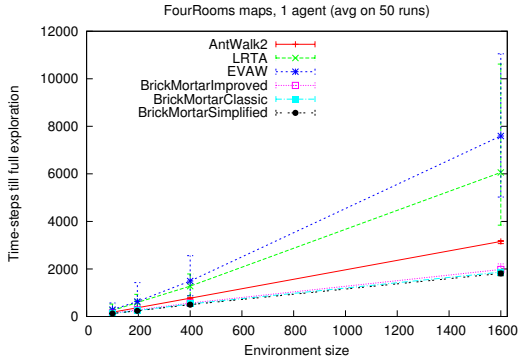
This gain in time before the identification of exploration termination is even more evident on the *Obstacles* map for the single agent case, when varying the size of the environment: the slope of Brick&Mortar is growing at almost 7 times the rate of the slope of BMI. The dispersion in the exploration time of Brick&Mortar (the difference in timesteps between the slowest and quickest run) is 80x the environment size, while the dispersion of BMI is **much lower**, at 6x the environment size (see figure 8(f)).

When fixing the size of the environment at 20x20 cells, and varying the number of agents on the *Obstacles* map, BMI requires from 84% less time for the single agent case, to 50% less time for 20 agents, to identify the exploration completion (see figure 8(e)).

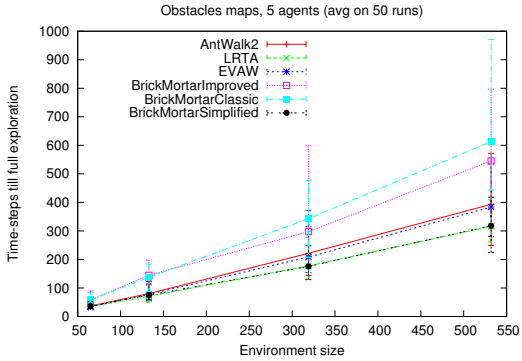
Figure 9 presents the overcost (in terms of time) brought by the identification of exploration completion, compared to a simple exploration of the environment. The overcost is calculated by substracting the time taken by BMS to simply explore the environment from the time taken by BMIS to identify the exploration completion.

C. Post-exploration rendez-vous

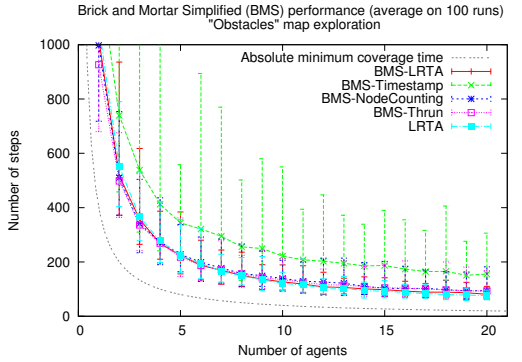
This section presents the performances of BMI, that gathers the agents at the end of their exploration. Intuitively, this capacity comes at a cost, generated by the time needed to return to the specified point. We should stress here that, with BMI, both identification of exploration completion and agents' rendez-vous occur at the same moment in time, when all the agents gather at the attractor cell. Therefore, the additional time that BMI needs to identify the exploration completion is bounded by the length of the longest return path (equal to the number of vertices in the diagonal of the environment), plus some variable cost induced by the time agents spend roaming through the return path, that would otherwise be closed. This overcost is clearly seen in environments without obstacles, as the *4 rooms* map (fig 7(b)). Despite this overcost, BMI does much better than Brick&Mortar on maps with lots of obstacles, due to its



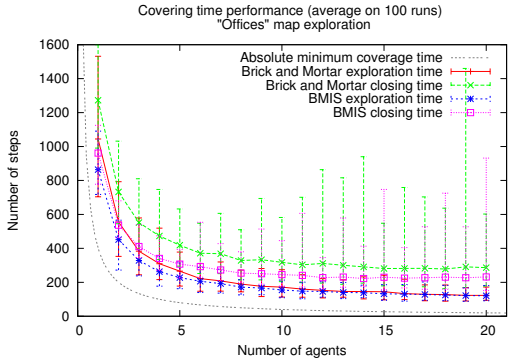
(a) Performances of analysed exploration algorithms, on maps without isolated obstacles (*Four rooms* maps used here). Brick&Mortar is by far the quickest explorer (lowest curve in this figure, almost overlayed with its variations, Brick&MortarSimplified and Brick&Mortar Improved). EVAW is an algorithm presented in [6], based on the VAW algorithm described in [20]. Ant-walk-2 is another algorithm described in [20].



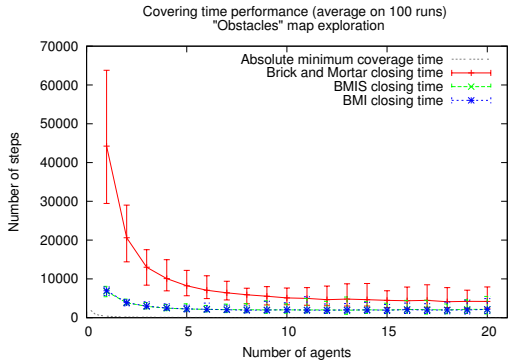
(b) Performances of analysed exploration algorithms. Although Brick&Mortar is the slowest algorithm on the *Obstacles* map, it is because it has the additional feature of detecting the exploration completion. Nevertheless, a dissected version of Brick&Mortar, BMS (Brick&Mortar Simplified), capable only of environment exploration (with no identification of exploration completion and which does not pay the cost of this additional feature), clearly places itself among the quickest algorithms (lowest curve in this figure, overlayed with LRTA).



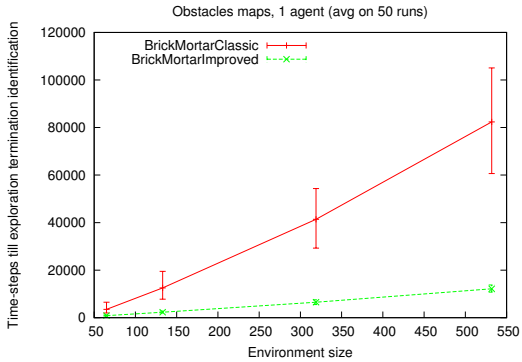
(c) Performances of Brick&Mortar Simplified variations on the *Obstacles* map. The BMS-Timestamp variation, that uses timestamps of last visit of cells as pheromone traces to disperse the agents (a version used by EVAW) takes between 1.5 and 2 times more steps to attain the exploration objective than all the other variations of BMS. It can thefore be discarded. The absolute minimum coverage time is calculated by dividing the surface of the environment by the total number of agents. Its interpolation is shown to easily compare the algorithms.



(d) Comparison of Brick&Mortar and BMIS performances. BMIS needs between 30% et 13% less steps to identify the termination of exploration on the *Office* map, compared to classic Brick&Mortar. The gain in exploration time of BMIS on the same map reaches 5-10% for 1 to 9 agents, after which the performances of the two algorithms equalize.



(e) Comparison of BMI and BM performances, *Obstacles* map. For the identification of exploration termination, BMIS requires from 84% less time for the single agent case, to 50% less time for 20 agents.



(f) Comparison of BMI and BM performances, *Obstacles* map. The slope of BM is growing at almost 7 times the rate of the slope of BMI. The dispersion of Brick&Mortar (the difference in timesteps between the slowest and quickest run) is 80x the environment size, while the dispersion of BMI is much lower, at 6x the environment size.

Fig. 8. Benchmarking results

optimised *loop closing* algorithm. We notice that, on the *Obstacles* map with size 26x26 and 532 explorable cells, in a setting with 5 agents, the time required by BMI to attain its exploration objective is on average 12% less than the time required by Brick&Mortar. (see figure 8(b)).

VII. Future work

The BMI algorithm described in this paper can be implemented on *sensor networks*. A suitable example could be a floor made out of *intelligent tiles* that can communicate and store data. BMI could provide the framework for cooperation between mobile robots and the surrounding environment, allowing them to use trails instead of vision to guide their navigation. Tiles equiped with scales, that can detect the presence of objects on their surface, could declare themselves as occupied with obstacles in the 2D map of the environment.

The algorithms described in this document are also applicable for the exploration of 3D environments, in the context of flying agents. Countrary to 2D environments, it is not possible anymore to leave traces in the environment. A solution might be to relax several constraints of ant algorithms: build and store a version of the map in the memory, authorize inter-agent communication and keep a communication link among the agents at any given moment of the exploration, in order to transmit data updates about the explored map. Under this hypothesis, as no markings are left in the physical environment, error-free localization of robots is required, to accurately communicate positions. In this sense, existing SLAM methods of high precision (3 cm) for interior environments could be employed.

The virtual marking system maintains its potential to indirectly solve navigation problems. Local marking and navigation rules are sufficient to navigate through the environment and allow the agents to avoid costly path calculations to the closest non-explored regions.

In addition, these algorithms are flexible enough to work with a varying agents' viewing range. The greater the viewing range of the agents is, the quicker they explore the environment.

VIII. Conclusion

This paper presents a new ant-algorithm for distributed exploration of unknown environments, known as Brick&Mortar Improved. Initially based on Brick&Mortar, an algorithm capable of identifying when environment exploration has ended, the new algorithm introduces a new feature for gathering the agents at a post-exploration rendez-vous point. This can make agents return to the entry of a building, once the exploration is complete, or procede to an evacuation point. It also improves several aspects of Brick&Mortar, sensibly reducing the time till identification of exploration completion. This new algorithm is then benchmarked against existing ant-algorithms on several widespread graph topologies.

Future work will imply the implementation of the new algorithm on a robotic platform, that would communicate with a grid-type environment, such as a network of intelligent tiles, or another type of smart surface.

References

[1] Antoine Bautin, Olivier Simonin, and François Charpillet. Minpos : A novel frontier allocation algorithm for multi-robot exploration. In *Intelligent Robotics and Applications*, volume 7507 of *Lecture Notes in Computer Science*, pages 496–508. Springer Berlin Heidelberg, 2012.

[2] Johann Borenstein, H. R. Everett, and Liqiang Feng. *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Natick, MA, USA, 1996.

[3] Ettore Ferranti, Niki Trigoni, and Mark Levene. Brick&Mortar: An online multiagent exploration algorithm. In *IEEE Int. Conf. Robotics and Automation (ICRA)*, 2007.

[4] Ettore Ferranti, Niki Trigoni, and Mark Levene. Rapid exploration of unknown areas through dynamic deployment of mobile and stationary sensor nodes. *Autonomous Agents and Multi-Agent Systems*, 19(2):210–243, October 2009.

[5] Antonio Franchi, Luigi Freda, Giuseppe Oriolo, and Marilena Vendittelli. A decentralized strategy for cooperative robot exploration. In *Proceedings of the 1st international conference on Robot communication and coordination*, RoboComm '07, pages 7:1–7:8, Piscataway, NJ, USA, 2007. IEEE Press.

[6] Arnaud Glad, Olivier Simonin, Olivier Buffet, and François Charpillet. Theoretical Study of ant-based Algorithms for Multi-Agent Patrolling. In *18th European Conference on Artificial Intelligence including Prestigious Applications of Intelligent Systems (PAIS 2008) - ECAI 2008*, pages 626–630, Patras, Grèce, 2008. M. Ghallab et al., IOS press.

[7] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[8] Sven Koenig, Boleslaw Szymanski, and Yaxin Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31:41–76, May 2001.

[9] L. Matignon, L. Jeanpierre, and A.-I. Mouaddib. Distributed value functions for multi-robot exploration. In *Proceedings of the International Conference on Robotics and Automation (ICRA'12)*, 2012.

[10] Nicolas Pepin, Olivier Simonin, and François Charpillet. Intelligent Tiles: Putting Situated Multi-Agents Models in Real World. In ACM AAAI, editor, *International Conference on Agents and Artificial Intelligence - ICAART'09*, Porto, Portugal, 2009.

[11] Anies Hannawati Purnamadjaja and R. Andrew Russell. Pheromone communication in a robot swarm: necrophoric bee behaviour and its replication. *Robotica*, 23(6):731–742, November 2005.

[12] R. Andrew Russell. Laying and sensing odor markings as a strategy for assisting mobile robot navigation tasks. *Robotics& Automation Magazine, IEEE*, 2:3–9, Sep 1995.

[13] R. Andrew Russell. Heat trails as short-lived navigational markers for mobile robots. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 3534–3539, Albuquerque, 1997.

[14] R.A. Russell, D. Thiel, and A. Mackay-Sim. Sensing odour trails for mobile robot navigation. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2672–2677, May 8-13 1994.

[15] Titus Sharpe and Barbara Webb. Simulated and situated models of chemical trail following in ants. In *Proceedings of the fifth international conference on simulation of adaptive behavior on From animals to animats 5*, pages 195–204, Cambridge, MA, USA, 1998. MIT Press.

[16] Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. In *ICDE'02*, pages 357–368, 2002.

[17] Olivier Simonin, Thomas Huraux, and François Charpillet. Interactive Surface for Bio-inspired Robotics, Re-examining Foraging Models. In *23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Boca Raton, États-Unis, November 2011. IEEE.

[18] Jonas Svennebring and Sven Koenig. Building terrain-covering ant robots: A feasibility study. *Auton. Robots*, 16(3):313–332, May 2004.

[19] Sebastian B. Thrun, (editors) David A. White, and Donald A. Sofge. The role of exploration in learning control, 1992.

[20] Israel A. Wagner, Michael Lindenbaum, and Alfred M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15:918–933, 1999.

[21] Xiaoming Zheng, Sven Koenig, David Kempe, and Sonal Jain. Multirobot forest coverage for weighted and unweighted terrain. *IEEE Transactions on Robotics*, 26(6):1018–1031, 2010.