Influence of Different Execution Models on Patrolling Ant Behaviors: from Agents to Robots

Arnaud Glad Olivier Simonin Olivier Buffet François Charpillet

MAIA Team, LORIA
INRIA / Nancy University
Nancy, France
firstname.lastname@loria.fr

ABSTRACT

Generally, swarm models and algorithms consider synchronous agents, i.e., they act simultaneously. This hypothesis does not fit multi-agent simulators nor robotic systems. In this paper, we consider such issues on a patrolling ant algorithm. We examine how different execution hypotheses influence self-organization capabilities and patrolling performances of this algorithm. We consider the mono and multi-agent cases, the synchronism and determinism hypotheses, and the execution of the model with real robots.

Categories and Subject Descriptors

I.2 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Experimentation, Theory

Keywords

Agent Cooperation::Biologically-inspired approaches and methods, Agent-based simulations::Simulation techniques, tools and environments

1. INTRODUCTION

When we talk about situated multi-agent systems, theoretical, simulation and robotic implementation aspects of the problem are usually taken separately, each under different hypotheses. In this context, do theoretical results always make sense in a robotic implementation? Fairly different hypotheses in a real implementation may lead to different system behaviors. Are we sure that we can investigate all the system properties through simulations? Simulations are not exhaustive, some undiscovered situations may lead to undesired behaviors.

Cite as: Influence of Different Execution Models on Patrolling Ant Behaviors: from Agents to Robots, A. Glad, O. Simonin, O. Buffet, F. Charpillet, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 1173-1180

Copyright © 2010, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

Whereas the multi-agent paradigm is implicitly built on the interactions between agents, works on swarm algorithms do not usually pay attention to the question of the execution model. Axtell [3] shows how the activation model can impact simulation results. Depending on the simulated model, changing the activation model leads either to statistically different results or qualitatively different and unrealistic outputs. The lack of information about the execution model may also lead to re-implementation errors and force to align the model on known experimental data [6]. But even this process does not ensure the faithfulness of the reimplementation as various execution models may produce the output on given settings.

In the sake of simplicity, swarm algorithms are usually considered as synchronous and do not specify how to solve conflicts between the agents. However, modelers often use asynchronous schedulers, either because of their unawareness of this simulation aspect or the efforts needed to adapt and implement models to synchronous execution models [17] (the designer has to distinguish the production of influences from the reaction of the world and to manage the synchronization and interactions between agents explicitly [7]). Also, even if more and more multi-agent simulators [13] implement synchronous schedulers, they often propose asynchronous schedulers by default.

When implementing a conceptual model, questions about execution models are often solved in relation to the constraints of the platform without taking care of the validity of these choices.

In this paper, we explore this question and the effects of the hypotheses made on the execution model of EVAP, an ant inspired approach for multi-agent patrolling. The patrolling task consists in deploying several agents in order to visit defined places of an environment as regularly as possible. It aims at gathering reliable information, seeking objects or watching over places to defend them against intrusions (building surveillance, fire hazard prevention, web pages indexation [1, 2, 4]). An efficient patrol requires to minimize the time between two consecutive visits of all the places of the environment.

Various approaches — as described in [1] — have been proposed to address this problem and are based on agents following Traveling Salesman Problem solutions, Ant Colony Optimization (ACO), stigmergic ant algorithms or reinforcement learning.

We are particularly interested in ant algorithms, proposed for the covering and patrolling problems by Wagner [16], which guarantee to repeatedly visit the environment with competitive performances. These algorithms are fully decentralized and rely on memoryless agents with very simple individual behaviors. Agents can only communicate through environment marking and, as they only mark and move according to their local perception, do not need information about the environment.

First, we introduce the patrolling problem and present the EVAP algorithm. In a second section, we review problems raised by the execution model (conflict resolution, scheduling for simulation and hypotheses for robotic implementation). Sections 4 and 5 are dedicated to the study of different execution models on two EVAP properties (respectively the patrolling property and the emergence of stable cycles). Finally, Section 6 concludes this work and presents some perspectives.

2. CONTEXT

2.1 The Patrolling Problem

As explained in the introduction, the multi-agent patrolling problem consists in visiting several places of a discrete environment in order to minimize the time between two consecutive visits. This delay is also called *idleness*.

2.2 Patrolling Ants

Ant-based approaches rely on simple agents evolving in a discrete environment. These agents are memoryless and do not have any environment representation. They are only able to move to neighboring cells, mark the current cell and perceive marks on neighboring cells.

The environment is represented as a strongly connected graph[1] G(V,E) and each vertex has a marking. In the general case, these markings can be seen as vector spaces, so that agents may mark vertices with multiple mark types.

Most ant algorithms fit this description and only differ on the order of execution of the possible actions and the semantics of the marking. We can cite, as examples, Vertex-Ant-Walk (VAW) algorithms [16, Appendix II-A], an exploration algorithm by Thrun [15], LRTA* and Node Counting [10], and EVAP/EVAW [9].

2.3 The EVAP Algorithm

The EVAP algorithm introduced in [5] is an ant inspired algorithm that only relies on the evaporation of a pheromone dropped by agents. Their behavior is defined as a gradient descent (Figure 1).

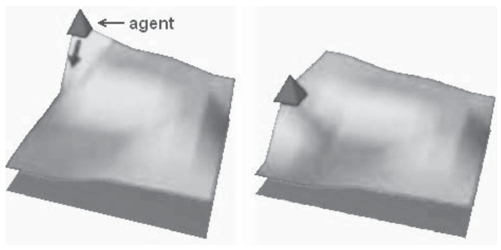


Figure 1: An EVAP agent descending a pheromone gradient

[1]In a strongly connected graph, a path exists between each pair of vertices.

Algorithm 1 describes an EVAP agent's individual behavior. When arriving on a vertex, the agent drops a fixed quantity Q_{max} of pheromones — marking its visit — and moves on a neighboring vertex according to its local perception. As the the pheromone evaporates (Alg. 2), the quantity remaining on a vertex represents the elapsed time since the last visit. So the agent behavior is defined by moving locally to the vertex that has not been visited for the longest time.

Algorithm 1: Behavior of an EVAP agent

```
while true do
    Find vertex $x$ of $Neighborhood$ with the lowest value
    (in case of a tie, make a random choice)
    Move to vertex $x$
    $Q_{pheromone}(x) \leftarrow Q_{Max}$
```

Algorithm 2: Environment Algorithm

```
foreach vertex $v \in Environment$ do
    $Q_{pheromone}(v) \leftarrow Q_{pheromone}(v).\rho$
```

It has been shown that EVAP has some interesting properties. In [9], we extended a proof of covering from [16]. i.e. the fact that all the vertices of the environment are visited in finite time. We will also see in Section 4 that the EVAP algorithm revisits all the cells of the environment infinitely often. This property is called patrolling property. This algorithm also exhibits a nice self-organization property. After a time, the algorithm converges to a cycling behavior. This is particularly interesting because cycles are known to be good solutions for the patrolling problem, in particular for Hamiltonian cycles which visit all the vertices of the graph exactly once in a tour and are therefore optimal solutions. Even when the algorithm has not yet converged, the patrolling performances are good and may even be pretty close to the optimum [5].

3. EXECUTION MODEL

At this point, do you feel yourself able to implement the EVAP algorithm exactly as we did? What about the scheduling or the resolution of conflicts between the agents?

Generally, works on reactive multi-agent systems only present the agents' behavior (like in Sec. 2.3), considering implementation details like conflicts resolution and scheduling hypotheses as implicit despite their significant influence on simulation and robotic results. These details are often chosen implicitly when implementing because of technical constraints which differ depending on the platform (robots, simulator, another simulator).

We focus here on these problems in the frame of discrete situated reactive multi-agent systems (i.e. algorithms intended to solve problems online in robotic applications).

3.1 Managing (non)Determinism

The first important point is to detail the handling of the non determinism in these models. Non determinism presents itself under three different aspects:

Hesitations.
Hesitations occur when agents have the choice between different possibilities. In the EVAP case, it happens on Alg. 1, line 2. This aspect is the easiest to handle as it is usually explicitly present in the algorithms.

Conflicts.
Conflicts come from agents' direct interactions when they have to share a resource. As these interactions are not present in the algorithm's description, conflict handling is rarely explicit. Let us take the EVAP example. What happens when two agents choose to visit the same cell ? (see Fig. 2-a) In such a conflict they may:

- visit the cell together (Fig. 2-b, this solution may be problematic in a robotic implementation and clearly suboptimal);
- solve the conflict with a priority mechanism (Fig. 2-c, e.g. first one arrived, priority based on their position — think about the priority rules at road junctions — or based on the agents identification — agents with lower IDs have priority over agents with higher IDs —);
- not move and so not solve the conflict, leading the system to a dead lock (Fig. 2-d).

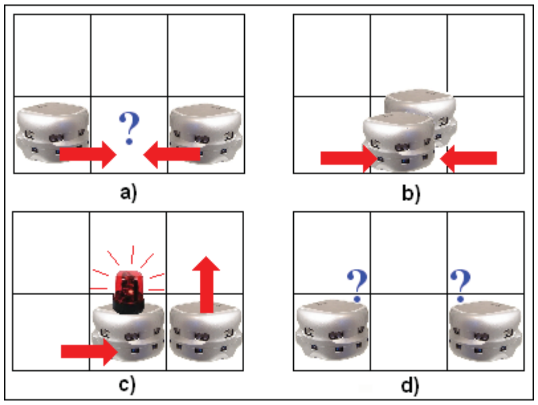


Figure 2: Agents conflicting to go on the central cell

We can see that the way conflicts are handled may change the global behavior of the system and that they are often solved because of implementation constraints.

Transitions and Time Granularity.
This is probably the most deceptive of these aspects. Simulations of discrete reactive MAS usually consider transitions between system states as discrete. Time is then considered as discrete — where an *iteration* is the atomic unit of time — but time granularity may greatly differ from one simulation to another (i.e. from milliseconds for robotic simulations to thousands of years for geological simulations). Conceptual models in swarm algorithms usually consider the duration of an iteration as the time needed for an agent to perform an atomic action (i.e. moving somewhere, picking up objects, ...). Problems arise when the durations of iterations in the conceptual model and in the implementation do not match.

Let us consider here the robotic implementation of a synchronous model where agents act exactly once per iteration. We can see that time granularity of the conceptual model (an iteration represents the time needed for all agents to perceive move and mark) and real robots (an iteration corresponds to a couple of milliseconds) do not match. There are two options to handle this problem.

- *Implementing the simulated system on robots as is:* If we want to stick to the model, agents should not be allowed to move again before all other agents have finished their action. All actions have to start and end within the same iteration. So, faster agents waste time waiting for slower ones when they are done, impacting the performances of the system. Moreover, this approach requires a synchronization point at the beginning of each iteration, which can be hard to implement and not desirable in a decentralized robot swarm.

- *Fitting the conceptual model to robots specificities:* In this case, agents make their move as soon as they are ready to. As the durations of various actions may differ (the duration of a given action may even differ from one timestep on another), the system inevitably drifts and soon loses its synchronization (i.e. an agent may start a new action before the others finished theirs). In this perspective, are the theoretical and simulation results based on the synchronous hypothesis still valid or are we studying a different model? For the answer, see Sec. 5.2.

3.2 Scheduling Hypotheses

Acting synchronously or sequentially may change the global behavior of the system but scheduling hypotheses are rarely given with the model. Moreover, despite the fact that ant algorithms are usually considered as synchronous — all agents act simultaneously —, discrete simulators rarely implement synchronous schedulers. In this section, we will see different scheduling models for discrete time simulation.

Synchronous Approaches.
Synchronous scheduling seems to be a clever choice. Direct interactions between agents (i.e. conflicts) only occur when two or more agents wish to do incompatible actions at the same time. So, synchronous simulation may be a useful tool to identify such conflicts and figure out how to solve them. However a synchronous scheduling is harder to implement than an asynchronous one. The influence-reaction model [7] explains how to approach this problem. First, all agents have to make their decisions without executing them (agents produce influences). In a second phase, the simulator has to combine influences and solve all the conflicts (reaction of the environment). There is no generic tool to solve conflicts automatically, so this task is left to the designer. Thus, although most discrete multi-agent simulators seem to be synchronous — because all agents make their move in a single iteration —, they implement in fact asynchronous schedulers.

Asynchronous Approaches.
Among all variations of asynchronous scheduling, we only present here asynchronous sequential scheduling (i.e. all agents act sequentially within an iteration).

- **Asynchronous cyclic scheduling**: At each iteration, agents are sequentially activated in a fixed order.
- **Asynchronous acyclic scheduling**: At each iteration, agents are sequentially activated in a random order.

Asynchronous sequential scheduling present the advantage — or disadvantage if the designer is not aware of it — of implicitly solving most of the conflicts between agents. The scheduling order makes agents act one after the other and thus are no longer in a conflicting situation. We can also notice that even if both cyclic and acyclic sequential schedulings solve conflicts, the first one solves them deterministically (when two given agents are in conflict, it is always the same one — the first in the list — who “wins”) while we cannot predict the solution for the second one.

3.3 Robotic Implementation

The implementation of swarm models with autonomous robots is a general and complex challenge. More precisely, the main issues are :

- Robots are autonomous, i.e. they are not synchronous when taking decisions. Moreover, using a scheduler does not make sense as robots hold their own processor which activates themselves. Although, robots can be synchronized, but in this case the system loses the property of being a decentralized system.
- Robots cannot be considered as identical agents. For instance two similar mechanisms may lead to two slightly different results. This implies that transitions cannot be considered as deterministic. Moreover, robots often evolve in a non deterministic world.
- Algorithms requiring active environments are difficult to implement due to the nature of the considered processes: marking the environment, dropping a pheromone! However, in these days, such mechanisms can be implemented via environments augmented with sensor networks [12].

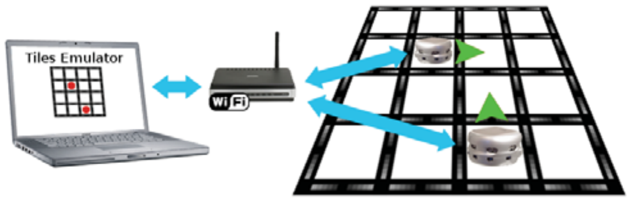


Figure 3: Schema of tiles emulator

In this paper, we analyze swarm algorithm hypotheses in order to take a step towards their robotic implementation. To examine the implementation of the EVAP algorithm with robots, we rely on the Tiles model [14] (see Fig. 3) which defines a device allowing the marking of the environment. This model makes the following assumptions:

- The environment is paved with square tiles, each one being embedded with an autonomous process able of simple computations and communications with a robot.
- Each tile is connected to its four neighboring tiles, defining a network of communication.
- Robots can locally read and write information on a tile, and diffuse information to others through the tiles.

We implemented the EVAP model on Khepera 3 robots and the environment marking/reading with a tiles emulator. The advantage of such a choice is that robots remain autonomous and are not synchronized with the active environment. A video of the EVAP model implementation on the tiles emulator is accessible at “http://www.loria.fr/∼simoniol/EVAP.html”. Results are analyzed in the following sections.

4. THE PATROLLING PROPERTY

We present here the patrolling property which consists in revisiting infinitely often and as regularly as possible all the vertices of the graph representing the environment. We first prove that agents never stop patrolling then we look at the performances of the simulated model with different numbers of agents, with various schedulers and conflict resolution mechanisms. Finally, we take a look on the problems raised by our robotic implementation.

4.1 Qualitative Results

It is not obvious — given the agent behavior presented in Alg. 1 — that all cells are revisited after a first coverage of the environment. This is a central problem. Even if simulation results show that agents actually patrol, nothing ensures that particular topologies may not lead to the formation of non revisited vertices islets.

It is easy to prove that EVAP ants cover the graph repeatedly. This can be done by assuming that at a given point agents stop patrolling the entire environment, leaving at least one vertex un(re)visited and the others visited infinitely often.

The agents continue to repeatedly visit the vertices of the patrolled area, each time reseting the pheromone value to Q_{max}. Therefore, after the covering of the patrolled zone, the values of the non-patrolled area cells are necessarily lower than any other cell. So, when an agent visits a vertex on the border of the patrolled area, it necessarily perceives a vertex of the non patrolled area which contains the minimum value of its perceptions. There is a contradiction with the initial assumption as, according to the EVAP agent behavior presented in Section 2.3, the agent will visit a cell of the non patrolled area.

This result is interesting as the demonstration is only based on the visit of a vertex and on the agents behavior. No assumption is made on scheduling, conflicts resolution or even the topology of the environment. Although having no hints about performance, we can be sure that the EVAP agents patrol no matter what hypotheses on the execution model are made.

4.2 Quantitative Results

Metrics.

In order to study EVAP in simulation, we use two performance criteria based on idleness taken from [11]:

- *Instantaneous Graph Idleness* (**IGI**) corresponds to the average idleness of all the vertices of the graph at a given moment,

- *Worst Graph Idleness* (**WGI**) corresponds to the worst idleness on the graph at a given moment.

In [5], the authors give theoretical bounds for the optimality of both IGI and WGI. Let c be the number of cells on the environment and n the number of agents. We have:

- an optimal WGI bound: $\frac{c}{n} - 1$,
- an optimal IGI bound: $\frac{\frac{c}{n}-1}{2}$.

Note that these bounds are only exact for Hamiltonian environments (i.e. environments that admit at least one Hamiltonian cycle). For other environments, these bounds cannot be reached and the actual optimal bounds are necessarily higher and depend on the topology.

Simulation Hypotheses.

As explained in Section 3, Algorithms 1 and 2 are not sufficient to simulate the system. We precise here hypotheses on the environment, how conflicts are handled and which schedulers are chosen.

- *Environment*: To represent the area to be patrolled, we choose to discretize the environment into a grid where each cell represents a small zone corresponding approximatively to the perception area of an agent. From each cell, agents can access their four neighbors. For coherence of agents perception, the evaporation process is computed on cells once per iteration once all the agents have moved.
- *Scheduler*: Asynchronous cyclic and acyclic sequential schedulers seem to be good choices. Indeed, because of the sequential actions, agents are never in conflict. Let us consider two agents in a conflicting situation where both of them choose to visit the same cell. According to Alg. 1 when an agent performed his move, it marks the cell he just arrived in. So when agent a_1 moves and marks the cell, agent a_2 will necessarily chose another destination. This scheduling type is particularly interesting to simulate robotic implementations where two robots may not be physically able to stand in the same location.
- *Transitions*: Transitions are constrained by the scheduling. We consider that each agent perceives, moves and marks once per iteration.

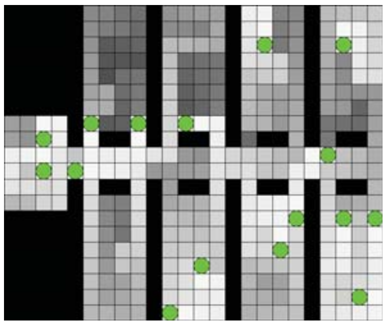


Figure 4: Agents patrolling on the "rooms" environment - 336 cells - non-hamiltonian environment

Simulation Experiments.

Except for the transitions, these hypotheses seem to fit well with the robotic implementation presented in Section 3.3. So, we can expect a quite accurate simulation of the robotic system.

We ran experiments on the "Rooms" environment (Fig.4), taken from [5], with 2 and 16 agents with *each scheduler*. Each setting has been run 50 times. Agents are randomly distributed through the environment at the beginning of each run. Figures 5 and 6 show idleness values averaged over 50 runs. One can see both that the scheduling does not change the algorithm's general behavior and has no significant impact on the performances.

We can here easily identify two distinct patrol phases on both figures (this is more visible on Fig. 6). The idleness — both IGI and WGI — increases to a high level at the beginning of the simulation before to decrease to a stable level. This is due to an exploration phase where agents mostly act randomly. At the initialization, all cell values are set to 0.

After the first coverage of the environment, agents perform better. The pheromone gradient created during the exploration phase leads them to a more efficient patrol. The algorithm shows then good performances in terms of IGI (which is quite close to the optimum) even if some cells may stay unvisited for a while, causing the WGI to raise.

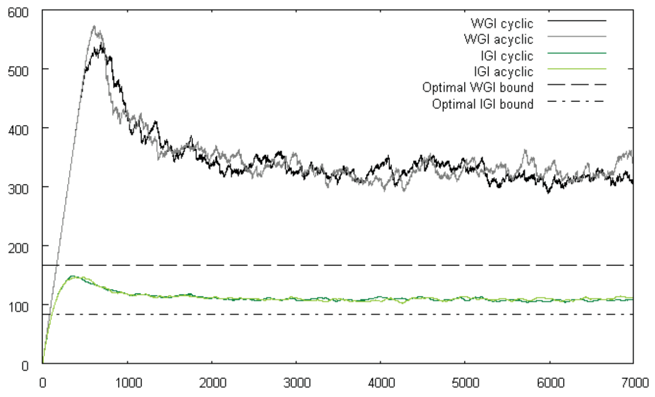


Figure 5: Idleness performance - 2 agents, rooms environment

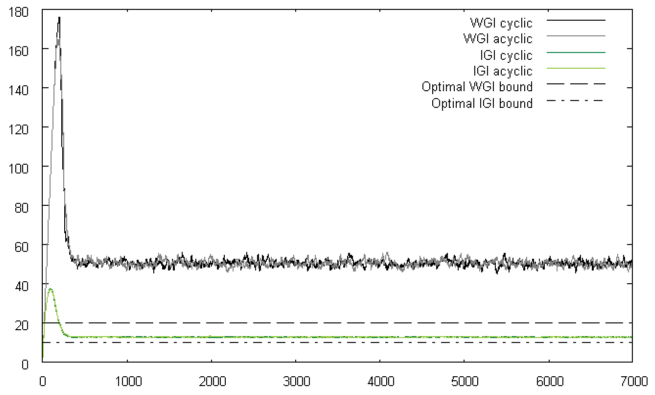


Figure 6: Idleness performance - 16 agents, rooms environment

Finally, we can note that the increase in the number of agents tends to smooth the curves and reduce the gap between the actual performances of the algorithm and the optimality bound.

Experiments with Robots.

A robotic implementation (Fig. 7) has been made with Khepera 3 robots and the tiles environment emulator presented in Section 3.3.

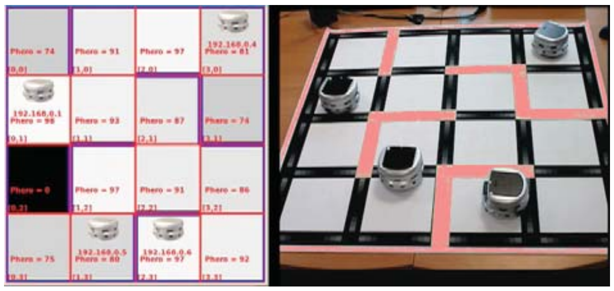


Figure 7: Robotic implementation of the EVAP algorithm on the tiles emulator

In order to avoid the case where two robots visit the same cell simultaneously, a reservation mechanism has been implemented. When an agent chooses its destination, it asks its current tile to reserve its destination tile and waits for an acknowledgement. If the tile is already reserved, the agent has to choose another destination or to wait if there is no other choice. This mechanism is very close to the conflict resolution of the *asynchronous acyclic sequential scheduler*.

Because agents are decentralized and do not wait for each other to perform their next action, the performances of the robotic system should, intuitively, be slightly better. The idleness measured in the robotic system have the same profile than the one of the synchronous simulation (see Fig. 5), the robotic implementation seems to be faithful to the EVAP model.

5. SELF-ORGANIZATION PROPERTY: CONVERGENCE TO CYCLES

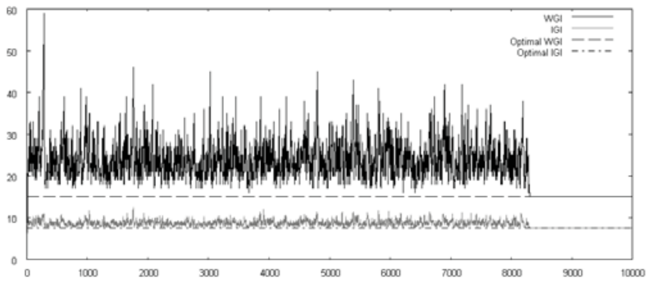


Figure 8: Idleness performance - 4 agents, 8x8 square grid environment

Fig. 8 shows the idleness for a single simulation experiment with 4 agents on an 8×8 grid without obstacles with an asynchronous cyclic sequential scheduler. We can notice that at approximately 8250 iterations, the WGI and IGI curves become flat and coincide with the optimal lines. This phenomenon is explicable by a self-organization property of the EVAP algorithm. It has been shown in [8] that the algorithm always converges to cycles which are generally good solutions for the patrolling problem, in particular in the case of Hamiltonian cycles that are always optimal.

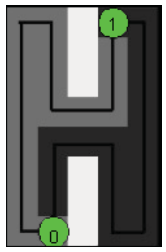
5.1 Known Results

Cycle Definition.

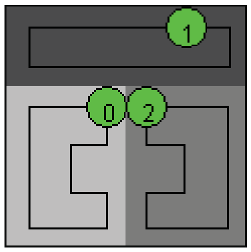
We consider here cycles from a system viewpoint. Cycles are defined as a finite sequence of system states where first and last states are equivalent system states.

A system state is defined by the respective positions of the agents and the pheromone gradient. Two system states are said equivalent when:

- agents positions are equals,
- pheromone gradients of the two states are equal.



a) a cycle with a common shared path



b) a cycle with three distinct paths

Figure 9: Two different types of cycles

The system may converge to different cycle types where each agent always covers the same distance as the others during a tour (see [9]). This property guarantees the quality of the solution in terms of idleness. Fig. 9 shows two types of cycles: a) a cycle where both agents follow the same trajectory, b) three agents, each one in its own path.

Proof of Convergence.

In [8] we showed that the EVAP algorithm always converges to a cyclic behavior. The proof is based on the fact that *EVAP operates over a finite state space.* If the system is (or is made) deterministic, the system converges necessarily to a cycle. If the system is non deterministic the system can be modeled as a Markov chain over a finite state space and necessarily converges to an absorbing graph (cycles being particular cases of absorbing graphs).

Absorbing graphs can be considered as cycles with some non deterministic points where agents are able to switch their position (i.e. agent a_1 patrols agent a_2's area and reciprocally) without breaking the cycle. Fig. 10 shows such an absorbing graph. Nodes of the graph represent the non deterministic points and edges all the (deterministic) simulation steps between them. This absorbing graph is optimal in term of idleness.

Cycles and Scheduling.

Unlike for the patrolling property, scheduling has visible effects on the convergence to cycles. The algorithm still converges to cycles but some of them may be unstable and break, depending on the scheduling model.

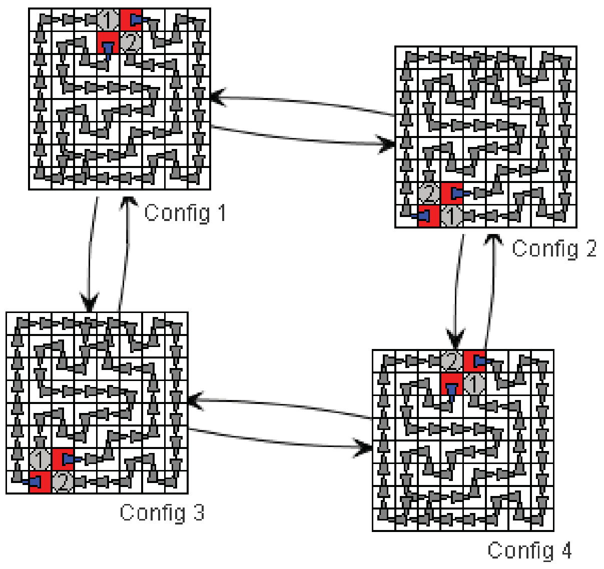


Figure 10: Example of absorbing graph with two agents - arrows on each cell represent the possible choices of the agent the last time the cell was visited

We take here the examples of asynchronous cyclic and acyclic schedulings.

Let Fig. 11 a) be the initial state of this example. Arrows on each cell represent the possible choices of the agent the last time the cell was visited and red arrows point towards the lowest pheromone values in the agents neighborhood, so to say, the cells that agents can visit in the next iteration.

Asynchronous cyclic scheduling: In this case, agents act always in the same order (according to their ID number). Agent a_1 has no other choice than going on the right cell (up and down cells were visited during the last iteration). Agent a_3 has then no other choice than moving to its right cell (which is the cell of lowest value in its neighborhood, see Fig. 11.b). The system will cycle between states a) and b).

Asynchronous acyclic scheduling: In this case, agents are activated in a random order. So, there is a chance that agent a_3 will act before agent a_1. In this case, a_3 has the choice to go either up or right. If it goes right, the cycle may persist temporarily (as in Fig. 11.b). If it goes up, agent $a1$ goes down and the cycle is broken (Fig. 11.c).

So, this kind of cycle is not stable with an asynchronous acyclic scheduling.

We have seen experimentally that for the same configuration (environment and number of agents), simulations with an asynchronous acyclic sequential scheduler do not produce the same variety of cycles and, as less possible cycles exist, take more time to converge. Although the quality of the solutions, in terms of idleness, does not depend on the scheduling process.

5.2 Cycling Robots?

We explained in the last section that the proof of convergence to cycles relies on a finite state space. However, when moving from an abstract model with identical agents to a real world with non-identical robots, one difference is that the moves are unlikely to have a constant duration. The speed of a move may depend on the robot and on the

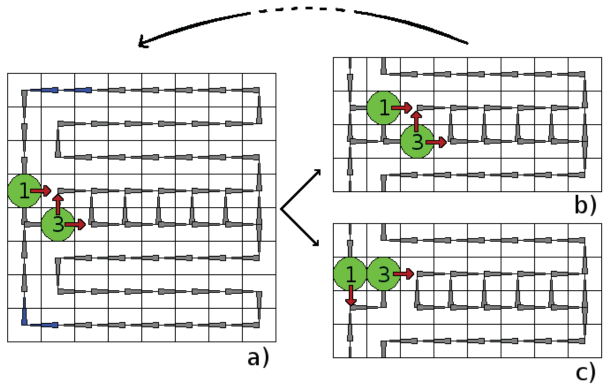


Figure 11: Example of absorbing graph with two agents. Red arrows show the agents' possible destinations.

type of move. We now show on a particular example that this difference leads to a system with an infinite number of states.

PROOF. Example: Let a_1 and a_2 be two agents on a 2×1 grid (the cells being denoted c_1 and c_2), both agents being initially located on c_1. Because of their ant behaviors, both will indefinitely alternate between c_1 and c_2.

Let us now assume that a_1 moves from one cell to the other in 1 time unit, and a_2 in $\sqrt{2}$ time units. Time can still be discretized by only considering time steps when a_1 or a_2 precisely arrives on a cell, i.e. time step of the form $t = k.1$ or $t = k * \sqrt{2}$, with $k \in \mathbb{N}$. But, because 1 is rational and $\sqrt{2}$ is irrational, each new time instant corresponds to a new position (in-between the cells) of either a_1 or a_2, and therefore to a new state.

From this, we can deduce that infinitely many new configurations of the pheromone field will be generated. So, although the system is completely deterministic, it will never exhibit a stable cycling behavior. □

Experiments with robots show that, on our implementation and with multiple robots, the system actually loses the ability to self-organize to stable cycles. However being able to converge to cycles, they may only persist for a short time before collapsing. The only solution we see to recover this property with multiple robots would be to introduce synchronization points so that the robotic system behaves like the simulations with an asynchronous sequential scheduler. Of course, as well as degrading performances — as faster robots have to wait for the other before moving again —, synchronization may not be desirable in a swarm of robots because of the recentralization it implies.

6. CONCLUSION

This paper studied the effects of different execution models on the behavior of the EVAP algorithm, a general ant-based approach for the patrolling problem. We have first presented aspects that are usually ignored when designing a discrete reactive multi-agent model:

- non determinism management (how agents react in case of multiple choices, conflict resolution policies, determinism of the transitions between states),

- hypotheses on scheduling (synchronous or asynchronous and discrete scheduling, simulation of continuous time).

These points are important because they may change the behavior of the model, gaining or losing properties depending on the choices made.

We also presented a robotic implementation relying on devices allowing the use of an active environment. Robotic systems come with their own hypotheses that are more restrictive than in simulation.

Unfortunately, these hypotheses are often determined by the implementation constraints. Most discrete multi-agent simulators only propose a default asynchronous scheduler resolving, by example, conflicting situations implicitly.

How comparable are a theoretical study, a simulation and a robotic implementation, each one coming with its own hypotheses/execution model? We studied two properties of the EVAP algorithm at these three levels and examined the effects of different execution models on them. While the patrolling property is resilient to the scheduling and to the way conflicts are solved, the convergence to cycles is impacted when these hypotheses change.

The problem comes for the conceptual models that are clearly incomplete. Choices have to be made on execution models, not when implementing but directly when defining the model so that theoretical works, simulations and robotic implementations work consistently among them. It would be interesting to formalize the completeness of discrete reactive multi-agent models in order to ensure that the objects studied in theory, simulation and robotics actually correspond to the same model.

7. ACKNOWLEDGEMENTS

The authors thank Romain Mauffray for his participation in the implementation of EVAP in the intelligent tiles and the experimentations with the robots.

8. REFERENCES

[1] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In *Advances in Artificial Intelligence — Seventeenth Brazilian Symposium on Artificial Intelligence (SBIA'04)*, pages 474–483. Springer-Verlag, 2004.

[2] R. Andrade, H. Macedo, G. Ramalho, and C. Ferraz. Distributed mobile autonomous agents in network management. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'01)*, 2001.

[3] R. Axtell. Effects of interaction topology and activation regime in several multi-agent systems. In *MABS 2000: Proceedings of the second international workshop on Multi-agent based simulation*, pages 33–48, Secaucus, NJ, USA, 2001. Springer-Verlag New York, Inc.

[4] J. Cho and M. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the International Conference on Management of Data (SIGMOD'00)*, 2000.

[5] H. Chu, A. Glad, O. Simonin, F. Sempé, A. Drogoul, and F. Charpillet. Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'07)*, pages 442–449, 2007.

[6] B. Edmonds and D. Hales. Replication, replication and replication: Some hard lessons from model alignment. *J. Artificial Societies and Social Simulation*, 6(4), 2003.

[7] J. Ferber and J. Muller. Influences and reaction : a model of situated multiagent systems. In *Proceedings of the 2nd International Conference on Multi-agent Systems*, pages 72–79, 1996.

[8] A. Glad, O. Buffet, O. Simonin, and F. Charpillet. Self-organization of patrolling-ant algorithms. In *CD-ROM Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems, SASO09*, 2009.

[9] A. Glad, O. Simonin, O. Buffet, and F. Charpillet. Theoretical study of ant-based algorithms for multi-agent patrolling. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08), pp. 626-630*, 2008.

[10] S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31(1–4), 2001.

[11] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul. Multi-agent patrolling: an empirical analysis of alternative architectures. In *Third International Workshop on Multi-Agent Based Simulation*, pages 155–170, 2002.

[12] M. Mamei and F. Zambonelli. Spreading pheromones in everyday environments through RFID technology. In *2nd IEEE Symposium on Swarm Intelligence*, 2005.

[13] F. Michel, J. Ferber, and O. Gutknecht. Generic simulation tools based on MAS organization. In *Proceedings of the 10th European Workshop on Modelling Autonomous Agents in a Multi Agent World MAMAAW*, 2001.

[14] N. Pépin, O. Simonin, and F. Charpillet. Intelligent tiles: Putting situated multi-agents models in real world. In *Proceedings of the International Conference on Agents and Artificial Intelligence*, 2009.

[15] S. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, 1992.

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

[17] D. Weyns and T. Holvoet. Model for simultaneous actions in situated multi-agent systems. In *First German Conference on Multi-Agent System Technologies*, pages 105–119. Springer–Verlag, 2003.