Self-Organization of Patrolling-Ant Algorithms

Arnaud Glad, Olivier Buffet, Olivier Simonin, François Charpillet
MAIA Team, LORIA
INRIA / Nancy University
Nancy, France
Email: firstname.lastname@loria.fr

***Abstract*—We consider here multi-agent patrolling as the task for a group of agents to repeatedly visit all the cells of a discrete environment. Wagner et al. [1] have introduced patrolling ant algorithms, where each agent can only mark and move according to its local perception of the environment. Among various results, it has been experimentally observed that for some algorithms the agents often self-organize in stable cycles which are near optimal in terms of visit frequency. This property is particularly interesting as it guarantees the long-term performance of the patrol.**

The present paper focuses on the convergence behavior of a typical ant-based algorithm, EVAW [1; 2]. The main contribution of this paper is to theoretically prove that the group of agents self-organizes in cycles under certain hypotheses. These hypotheses rely on some implementation details that allow to control the predictability of the system. In addition to these qualitative results on the convergence behavior, we aim at experimentally evaluating its characteristics. This led us to a second contribution: an algorithm that detects steady states. Finally, we propose an improved behavior that dramatically speeds up the self-organization and allows us to experiment on larger problems (both in terms of size and number of agents).

I. Introduction

Efficient exploration and surveillance of large environments is a central issue in mobile robotics, but multi-agent patrolling algorithms can also be used in other applications like the surveillance of a network or of web sites [3; 4]. Various approaches to multi-agent patrolling have been proposed —as described and evaluated in [5]— and based for example on agents i) following the solution of a Traveling Salesman Problem, ii) applying heuristic rules, iii) negotiating objectives or iv) learning by reinforcement. The performance of patrolling algorithms is usually measured through the average idleness over the environment (time between two consecutive visits of a cell).

In this context we are particularly interested in ant covering and patrolling algorithms, introduced by Wagner et al. [1], which guarantee to repeatedly visit all the cells of a discrete environment and with competitive performances in terms of idleness [6; 7].[1] These algorithms do not require information about the environment, as each agent only marks and moves according to its local perception of the environment.

In this paper, we do not focus on the performance of the patrol in terms of idleness, but study its long-term behavior. Indeed, it has been observed that the system often self-organizes in stable cycles. These cycles, which are Hamiltonian or quasi-Hamiltonian, usually provide a near-optimal solution to the multi-agent patrolling problem. This is for example the case for various algorithms in the VAW family [1] and in particular in EVAW [2], the typical ant-based algorithm we consider here, and which is described in Section II.

The main contribution of this paper, presented in Section III, is to theoretically prove that the group of agents self-organizes in cycles under certain hypotheses. These hypotheses, linked to the determinism of agents and their interactions, allow to control the predictability of the system. In addition to these qualitative results on the convergence behavior, we experimentally evaluate its characteristics. To conduct these experiments, we have to introduce an algorithm that detects steady states, which constitutes a second contribution (see Section IV). Finally, we propose in Section V an improved behavior that dramatically speeds up the self-organization and allows us to experiment (Section VI) on larger problems (both in terms of size and number of agents). Finally, Section VII concludes on this work and presents its perspectives.

II. Background

A. Patrolling Problem

Here, the discrete (and finite) environment $\mathcal{E}$ is modeled as a directed strongly connected graph[2] where vertices are cells $c_j \in C$ (neighborhood $N(c_j)$). The *state* of an ant agent $a_i \in A$ is described by its current cell: $s(a_i) = c_j$. In this context, we consider multi-agent patrolling as the problem for the agents to repeatedly visit each cell in the environment.

B. Ant-based Approaches

In a typical ant-based approach, the ant agents do not know their environment and are memoryless, and each cell

[1]We do not know of any work comparing ant algorithms (also called real-time search algorithms by Koenig et al. [6]) and other types of patrolling algorithms [5] in the literature.

[2]A graph is strongly connected if a path exists between any two vertices.

c_j has a marking $m(c_j)$ ($\in M$ the set of possible markings). The agents can move to a neighboring cell, mark their current cell and perceive the marking of neighboring cells. This marking of the environment $\mathcal{E}$ is inspired by real-world ant pheromones. From this, we can define:

- the *state* of the *environment* $s(\mathcal{E})$ as the marking of all cells;
- the *state* of the *system* as the tuple $s = (s(\mathcal{E}), s(a_1), \cdots, s(a_{|A|}))$.

When M is a vector space (e.g. $M = \mathbb{R}^n$), two environment states $s(\mathcal{E})$ and $s'(\mathcal{E})$ are said to be *equivalent* iff their markings (m and m') differ by a constant:

$$\exists k \in M,\ \forall c_j \in C,\ m(c_j) - m'(c_j) = k.$$

When M is one-dimensional ($n = 1$), we often identify an environment state $s(\mathcal{E})$ with the equivalent state $s^*(\mathcal{E})$ whose minimum cell marking is 0 ($\min_j m(c_j) = 0$). The same notion of equivalence naturally extends to system states. Plus, when necessary, an index t is added to states to specify the time-step, e.g. $s_t(a_i) = c_j$.

This model fits a variety of ant-based approaches like: some Vertex-Ant-Walk (VAW) algorithms [1, Appendix II-A], an exploration algorithm by Thrun [8], LRTA* and Node Counting [6], and EVAP/EVAW [2]. They mainly differ depending on the semantics of the marking.

C. EVAW

This paper focuses on the EVAW algorithm [2] because it is typical of ant patrolling and its formalism is convenient to derive theoretical results.

VAW/EVAW's marking scheme is characterized by: i) $M = \mathbb{N}$ and ii) an agent a_i marks its current cell by setting $m(s(a_i)) \leftarrow t$ (the current time step). With this marking scheme, up to $|A|!$ system states map to the same environment state (this upper-bound is not reached when multiple agents are on the same cell). As detailed in Algorithm 1 and as illustrated by Figure 1, at each time step an EVAW agent simply 1) moves to the neighboring cell with the smallest marking and 2) marks this cell with the current time t.

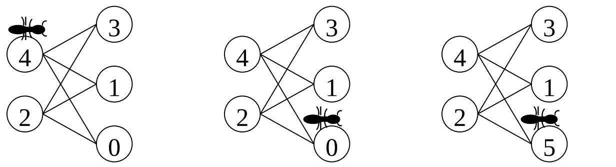
Algorithm 1: The EVAW algorithm executed by one agent

```
1 $c \leftarrow$ random cell in $C$ ;   /* initialization */
2 forall $t = 1, \cdots, \infty$ do
3     $c \leftarrow \arg\min_{d \in N(c)} m(d)$ ;   /* descent */
4     $m(c) \leftarrow t$ ;   /* marking */
```

This first algorithm is very simple, but does not specify how multiple agents running in parallel interact together. We make here the assumption that one agent observes its vicinity and moves without any other agent acting in the mean time, which is rather realistic compared to an implementation on robots (this prevents two agents from ending up on the same cell). Alg. 2 gives a sequential version of the multi-agent algorithm. One can observe that two lines involve non-deterministic choices:

- Line 3 – **Conflicts**: A *conflict* occurs when multiple agents "would like" to move to the same cell but the result depends on who moves first, as in Fig. 2-a.
- Line 4 – **Hesitations**: An agent a_i *hesitates* when it encounters choices between two cells which were last visited simultaneously (by two different agents) as in Figure 2-b.

In practice, it is often the case that conflicts and hesitations occur simultaneously.



a) current state b) descent (line 3) c) marking (line 4)

Figure 1. One iteration of the EVAW algorithm (Alg. 1)

Algorithm 2: Sequential EVAW for multiple agents

```
1 forall $i \in \{1, \cdots, |A|\}$ do  $c_i \leftarrow$ random cell in $C$
2 forall $t = 1, \cdots, \infty$ do
3     forall $i \in \{1, \cdots, |A|\}$ do
4         $c_i \leftarrow \arg\min_{d \in N(c_i)} m(d)$
5         $m(c_i) \leftarrow t$
```

0	1	**[2]**
0	0	0
0	1	**[2]**

a- Two conflicting agents

1	[8]	7	**[8]**
2	5	6	7
3	4	5	6

b- One hesitating agent

Figure 2. Two system states corresponding to uncertain situations (agent positions are represented by brackets): a) both agents would like to move to the '0' cell between them; b) the right agent has a choice between two '7' cells: the one on its left —last visited by the other agent— and the one below it —last visited by the right agent itself.

In a simulation, another way to handle interactions between agents is to 1) move all agents, then 2) let all agents mark their cells (see Algorithm 3). This "synchronous" version of EVAW (all-agents-move before all-agents-mark) is conflict-free, but does not model properly what would happen with robots because two agents are likely to be attracted by the same cell. In such a case, the two agents will keep making the same choices for some time (until the next hesitation at least), following the same path, which will result in a bad exploratory behavior.

Algorithm 3: Synchronous centralized EVAW for multiple agents

```
1 forall $i \in \{1, \cdots, |A|\}$ do  $c_i \leftarrow$ random cell in $C$
2 forall $t = 1, \cdots, \infty$ do
3     forall $i \in \{1, \cdots, |A|\}$ do
4         $c_i \leftarrow \arg\min_{d \in N(c_i)} m(d)$
5     forall $i \in \{1, \cdots, |A|\}$ do  $m(c_i) \leftarrow t$
```

D. Some Known Results

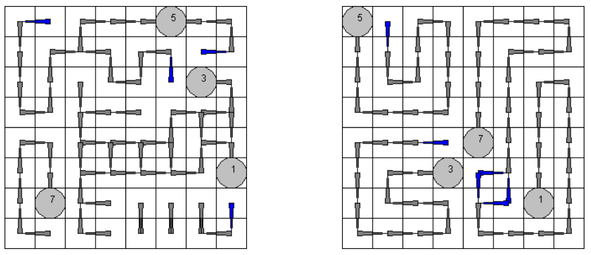
For each of the algorithms mentioned in Sec. II-B it has been proven that a group of ants repeatedly visits each cell in its environment, i.e. performs the patrolling task [1; 6]. To our knowledge, an upper-bound on the covering time is known for each of these algorithms except Node Counting. In the case of EVAW, we denote this bound by T_{vis}.

Experiments show that these algorithms exhibit good performances in terms of idleness [6; 7], although worst case performances of some of them (e.g. Node Counting) can be very bad — typically on complex hand-made topologies.

It is also known that the original VAW algorithm, EVAP and EVAW are quasi-equivalent and may converge to stable cycles – after an exploratory phase (Fig. 3-a) – where each agent repeats a sequence of cells infinitely often (Fig. 3-b). Only limited results are known about these cycles, e.g.: i) in the single-agent case an agent that follows an Hamiltonian cycle[3] once will repeat it forever [1]; ii) in the multi-agent case agents on separate cycles have equal-length cycles [2].

III. Convergence Results

In this section, we characterize the long-term behavior of the EVAW algorithm. This study of the convergence of the EVAW algorithm follows a different approach from previous work as, rather than looking at cycles from the agents' viewpoint, we consider cycles from the environment's viewpoint. Indeed, from now on a *cycle* ζ is defined as a sequence of *system* states that repeats forever.



a) exploration phase   b) agents reached a cycle

Figure 3. Pheromone field before and after convergence to a cycle (Arrows represent the direction followed by the agents at the time of the last visit of a cell)

[3]An Hamiltonian cycle visits each vertex of the graph exactly once.

A. Single-Agent Case

Theorem 3.1 (Convergence in the Single-Agent Case): Considering an environment $\mathcal{E}$ and a single agent a, EVAW converges to a cycle in finite time.

Proof: The proof comes from two facts: 1) the set of reachable system states is bounded; and 2) the agent's behavior becomes deterministic in finite time.

1) For any cell c, the time between two successive visits of c is bounded by T_{vis} (as explained in Sec. II-D). With "equivalent" states $s^*(\mathcal{E})$, this leads to: $\forall c \in C,\ 0 \leq m(c) \leq T_{vis}$. The set of reachable environment states is therefore finite since it can be bounded from above by $(T_{vis}+1)^{|C|}$. The same is true for system states since, with a single agent, there is a one-to-one mapping between environment and system states.

2) Agent a cannot visit — and therefore mark — two cells at the same time. Thus, two cells have the same marking if and only if this marking is 0 (i.e. they have not been visited yet). So, as soon as the agent has covered the environment — after at most T_{vis} steps — the agent cannot hesitate anymore: its behavior becomes deterministic.

So, after time step T_{vis}, EVAW deterministically generates a recurrent sequence of system states ($s_{t+1} = f(s_t)$). Because there is a finite number of reachable system states, this sequence converges to a cycle in finite time. ■

B. Multi-Agent Case

In the multi-agent case, we still have that all cells are visited infinitely often. An upper bound on the number of reachable system states is here $|A|! \times (T_{vis}+1)^{|C|}$.

Yet, having visited all cells once does not mean the system's dynamics are now deterministic. Both conflicts and hesitations can still be encountered. These sources of non-determinism can be tackled in various ways:

- **Conflicts:** One may specify a deterministic protocol deciding in which order agents act. This protocol may depend for example on agents' ID numbers or on their relative position in the environment.
- **Hesitations:** Here, one may pick a cell depending on its direction (e.g. an agent may prefer not turning) or depending on which agent left the last marking (if this information is available, an agent may prefer its own trace).

How decisions are made deterministic will of course influence the patrolling behavior. The proof of the following theorem is immediate as it essentially relies on the observation that the system can be modelled as a Markov chain.

Theorem 3.2 (Convergence in the Multi-Agent Case): The theorem is slightly different depending on the behavior — deterministic or not — of the agents.

- If the system is made deterministic, it converges to a cycle in finite time (same hypothesis and similar proof to theorem 3.1).

- If the system is not made deterministic, its behavior can be modeled as a Markov chain over a finite state space (some background about Markov chains is provided in Appendix A). It therefore necessarily ends up in a recurrent subgraph, cycles being particular cases of such subgraphs (see App. A2).

These two theorems confirm the observations of cycles and clarify the behavior of EVAW agents. Yet, we do not know for example how fast cycles or recurrent subgraphs are reached, or how many states they include. In the absence of theoretical results on such points, we look in this paper for experimental results.

IV. Cycle Detection

A key issue when studying the convergence of patrolling behaviors experimentally is the ability to detect cycles or recurrent graphs.

There exist multiple cycle detection algorithms for deterministic systems, such as the tortoise and the hare algorithm by [9] or Brent's algorithm [10]. This section presents a novel algorithm — extending the tortoise and the hare — we introduce for the detection of stable constructions in non-deterministic cases.

A. The Tortoise and the Hare [9]

Let us consider a set X, a function $f : X \mapsto X$ and a sequence u defined by $u_0 \in X$ and, for all $n > 1$, $u_{n+1} = f(u_n)$.

The cycle-detection algorithm proposed by Floyd [9] [11, exercises 6-7, page 7] detects whether the sequence u ends up on a cycle by comparing the evolution of u (called "the tortoise") and v defined by $v_n = u_{2n}$ (called "the hare"). As shown on Algorithm 4, a cycle is detected when $u_n = v_n$, $n > 0$ (i.e. when the hare and the tortoise meet at the same point).

Algorithm 4: The Tortoise and the Hare Algorithm

```
1 $v \leftarrow u$
2 repeat
3     $u \leftarrow f(u)$
4     $v \leftarrow f(f(v))$
5 until $u = v$
```

B. Handling Uncertain Dynamics

1) Difficulties: We now consider the use of the tortoise and the hare cycle-detection algorithm to detect whether a group of EVAW agents have reached a convergence point. A first important point is that both the hare and the tortoise should evolve identically, making the same random choices. This requires that they both rely on the same sequence of pseudo-random numbers.

Considering "determinized" versions of EVAW, a cycle may occur only after the coverage phase, so that Alg. 4 can be used flawlessly.

Otherwise, if non-deterministic transitions remain, the algorithm will stop in case:

- of a true cycle, as expected (see Fig. 3-b),
- of a recurrent subgraph, if the algorithm does not notice that non-deterministic transitions have occurred between two visits of the same state (see Fig. 4),
- of a transient subgraph, if the algorithm does not notice that some non-deterministic transitions will get the system out of this subgraph once and for all (see Fig. 5).

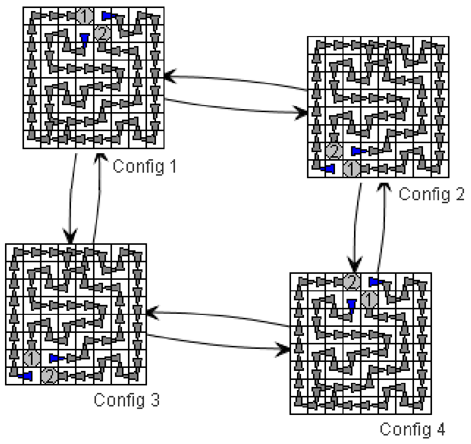


Figure 4. An example of a simple recurrent subgraph with two agents and four non-deterministic nodes

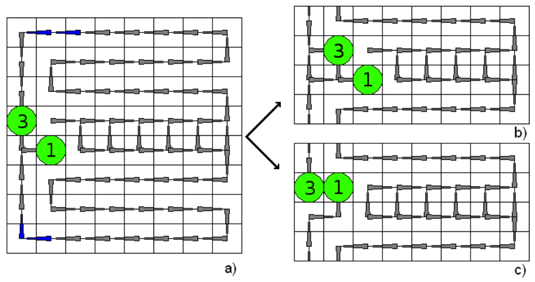


Figure 5. An example of a transient subgraph. a) Agent one is "hesitating" between up and right directions. b) While the agents keeps going right, the system seems to cycle; c) if the agent choose the up direction, the transient subgraph breaks.

If true cycles and recurrent subgraphs correspond to actual cycling behaviors, transient subgraphs can be considered as false-positives and therefore should not make the detection algorithm stop.

Checking if what has been detected is actually a true cycle requires simulating the system until the same state is reached without having any random choice to make. If it is not the case, we have a fake cycle which is either a transient subgraph or a recurrent subgraph. The following section explains how our algorithm distinguishes the two types of "fake" cycles (recurrent and transient subgraphs).

2) Our Detection Algorithm: Making the difference between a recurrent subgraph and a transient phenomenon is more difficult. Expanding the whole subgraph (i.e. all possible futures) is a long and memory hungry process. Since the objective of the detection is to analyze multiple runs of the algorithm, we just need to keep on simulating the system until a recurrent subgraph (possibly a cycle) is reached.

This is achieved by incrementally constructing the subgraph visited after a "fake cycle" has been encountered. We define a node of the graph as a *system state* where at least one agent has to make a decision, i.e. a state followed by a non-deterministic transitions. Each edge of the graph represents all the simulation steps (states followed by deterministic transitions) between two nodes.

When a node is detected (i.e. a conflict or an hesitation occurs), the algorithm computes and stores all its children. Then, the next node to be visited is chosen, among these children, by following the agents' behavior. At regular intervals, the algorithm analyzes the current subgraph to check if the system entered a cycle or a recurrent graph. To avoid memory overflows, a deadline is fixed that stops the graph construction (and erases the graph) when a maximum number of nodes have been visited. The simulation then continues — and the cycle detection *via* the tortoise and the hare algorithm is restarted — from this point.

V. Improved EVAW

This section is motivated by the combinatorial explosion of the convergence time with respect to the environment size and the number of agents. In order to propose an improved algorithm, we first introduce an appropriate visualization tool allowing us to analyze the path formation. From this analysis, we design a new heuristic which relies on detecting singular patterns.

A. Paths Visualization

The numerical marks left by the ants form a potential field, so that an intuitive idea is to visualize the associated gradient vector field, i.e. to draw, from each cell c, an arrow in the steepest descent directions (i.e. to each cell in $\arg\min_{d \in N(c)} m(d)$). Yet, as illustrated by Fig. 6-a, paths are not visible on this representation. We therefore propose a new representation that fixes this problem: at each time step we draw an arrow from a cell c to the neighboring cells reached by following the *flattest* ascent directions (i.e. the cells in $\arg\min_{d \in N(c) s.t. m(d) \geq m(c)} m(d)$). As illustrated in Fig. 6-b, the result clearly exhibits paths and Hamiltonian cycles when they appear. The reason for this result is that the flattest ascent direction usually represents the path followed by the last agent that has visited the cell.

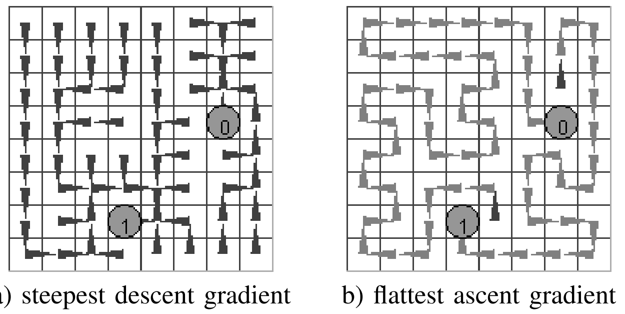


Figure 6. Two vector-field representations

B. Observation

We can see experimentally, and it is more visible on larger environments, that EVAW builds cycles from partial Hamiltonian paths appearing during the convergence process. We focus on cells which are local minima ($m(tail) \leq \min_{c \in N(tail)} m(c)$) and which we call *tails* since they correspond to most of the path beginnings we observe.

An interesting property is that, when an agent enters an existing path by its tail (dark cell on Fig. 6-b), it may follow the path to its end without reorganizing all the pheromone field, thus favoring convergence to cycles. Because of their limited knowledge of the environment (perception limited to the neighboring cells, no memory), agents are not able to identify paths and:

- do not always enter a path by its tail when possible,
- destroy paths when not entering them by their tails.

If identifying paths is difficult, an agent can easily detect a tail among its neighbors using an extended perception area (13 cells instead of 4 in our examples). Indeed, in order to detect this pattern an agent only needs to see, in addition to its current perception, the neighborhood of its adjacent cells (i.e. all the cells within a Manhattan distance of 2) as illustrated by Fig. 7.

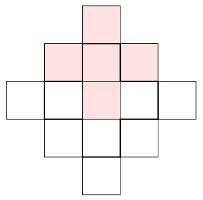

The grey cross represents the vicinity used to determine whether the cell to the North of the agent is a tail or not.

Figure 7. Perception area of an EVAW+ agent on a grid

C. Speeding Up Cycle Formation

Based on these observations and on the ability to detect tails, we extend EVAW by forcing the agents to move to a tail whenever one is detected. This variant — not limited to grid environments — is called EVAW+ and is shown on Alg. 5, where $tails(N(c))$ is the set of neighboring cells of c which match the tail pattern.

Algorithm 5: An enhanced version of EVAW

```
1 $c \leftarrow$ random cell in $C$
2 forall $t = 1, \cdots, \infty$ do
3     if $tails(N(c)) \neq \emptyset$ then
4         $c \leftarrow \arg\min_{d \in tails(N(c))} m(d)$
5     else
6         $c \leftarrow \arg\min_{d \in N(c)} m(d)$
7     $m(c) \leftarrow t$
```

D. How to Maintain the Patrolling Property

Although this improvement of the EVAW algorithm speeds up convergence (see Section VI), it leads to the loss of the patrolling property. Indeed we have observed rare cases where, after some time, some cells of the environment are not visited anymore. One can see on Fig. 8 a), c) and e) that the agent has no choice — according to Alg. 5 — but to follow path tails (darker arrows). As a consequence, the upper left cell and its two neighbors will not be visited anymore.

This bug can be fixed easily by adding a higher priority rule saying that, if the difference between the value of the candidate tail and the value of the oldest neighboring cell is greater than a chosen threshold T_{thr} (for example $10 \times |C|$ to avoid inhibiting EVAW+'s heuristic), then the agent should move to this oldest neighbouring cell. With this modification, we can prove again that EVAW+ agents cover their environment in bounded time [1, Th. 3, App. II-A], using $\Delta = T_{thr}$. From this point, we immediately have that the theoretical results presented for EVAW in this paper also hold for EVAW+.

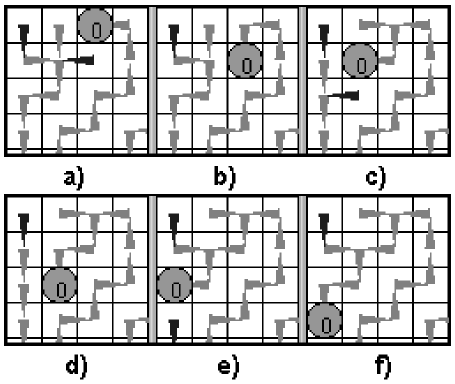


Figure 8. Loss of the patrolling property

E. Another Interesting Pattern

Tails are a first type of interesting pattern that appears when EVAW ants are patrolling. When the paths followed by the ants are getting stable, one can observe that the outgoing direction of some cells remains unstable. We call this phenomenon a *flip-flop*. Fig. 9 shows one ant following a non-Hamiltonian cycle with two flip-flops. In the right flip-flop (located on the cell whose value moves from 6 to 14), the outgoing arrow will alternate between the up and left directions.

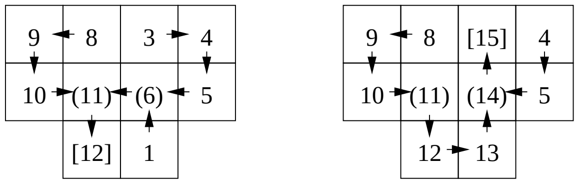


Figure 9. One ant (noted by brackets) following a non-Hamiltonian cycle with two flip-flops (noted by parentheses). The figures show time steps 12 and 15.

In a non-Hamiltonian environment, flip-flops are often useful. They make it possible to visit some cells more often than on others. A consequence is that it may not be a good idea to try to get rid of a flip-flop when in a non-Hamiltonian environment: another flip-flop will appear somewhere else.

We currently envision the idea of experimenting with a version of EVAW integrating a flip-flop eraser. An objective could be to reduce the number of flip-flops to a minimum value (and stop chasing this pattern when the number of detected flip-flops becomes stable).

VI. Experiments

This section presents experiments conducted to illustrate the theoretical results presented in Section III and study the convergence behavior of the system quantitatively, measuring for example the time to converge and the probability to converge to cycles. We look at the effect of a growing number of agents and of a growing environment size, hence the use of a scalable grid topology. To our knowledge no such experiments have been conducted with other patrolling ant algorithm to date.

A. Experimental Setting

We need to detail how conflicts and hesitations are handled in the implementations of EVAW and EVAW+ used in our experiments.

First, our early implementations of EVAW were relying on the Madkit/turtlekit framework [12] whose default behavior is to always schedule the agents in the same fixed order (an asynchronous sequential cyclic scheduling). We have kept this scheduling process since then (EVAW being now implemented in JAVA without Madkit/turtlekit), so that conflicts are solved with agent being prioritized according to their ordering.

Then, depending on how they behave in case of hesitations, we differentiate two types of agents: 1) non-deterministic agents that act randomly and 2) deterministic agents that prefer moving right rather than backward, left and finally forward. The former are used by default in our experiments (to observe the effect of this non-determinism) while the latter are only employed to verify some specific theoretical results.

Most experiments are characterized by the tuple (n, s, α), where n is the number of agents, $s \times s$ is the size of an empty square grid and α denotes the algorithm ('-'= EVAW, 'd'= deterministic EVAW, '+'= EVAW+). At least 1300 (and up to 10 000) simulation runs of at most 3×10^6 iterations were used. Agents were initially placed randomly.

B. Results

We ran experiments with a variety of settings ($n \in \{1, 2, 3, 4, 5, 6, 8, 10\}$, $s \in \{8, 14\}$, $\alpha \in \{-, +, d\}$). Tables I to III provide part of the collected statistics.

1) Convergence Behavior: Our first aim was to verify the convergence results of Sec. III using four settings: $(1, 8, -)$, $(1, 8, d)$, $(3, 8, -)$ and $(3, 8, d)$. As expected, the statistics gathered in Table II show that the only case where the convergence to true cycles is not guaranteed (i.e. when "recurrent graphs" are found) is with multiple non-deterministic agents ($(3, 8, +)$ and $(3, 8, -)$).

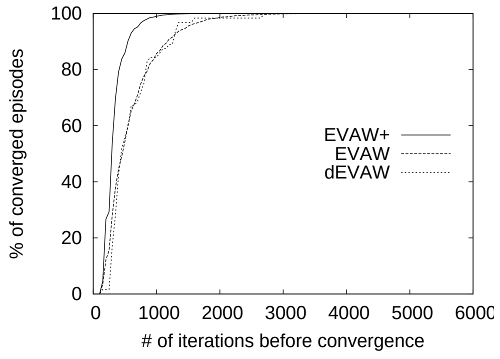
2) EVAW vs EVAW+: Then, let us consider the qualitative results gathered in various experimental settings — shown in Table II — plus Figures 10, 11 and 12, which present convergence curves (the probability to have converged to a cycle after N iterations) only taking cycles into account, so that they do not always reach 100%.

One can observe that EVAW+ always converges faster than EVAW, the task being more difficult when more agents or larger environments are involved. In fact, it would not be practical to experiment with more complex settings using the original EVAW algorithm.

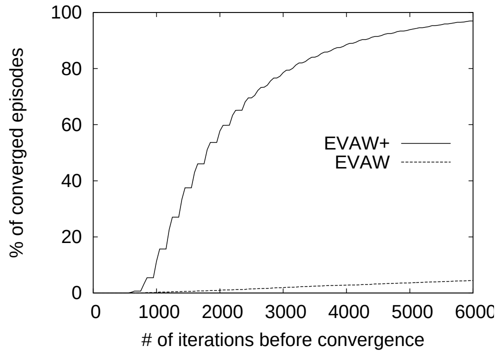
Shape of the Convergence Curves: Looking at the various convergence curves, as could be expected they all look like cumulative distribution functions (CDF) of exponential distributions. This means that, on average, the probability to converge is the same at each time step. They differ from these CDFs 1) at the beginning — during an initial exploration phase — and 2) by the fact that "steps" appear (especially in the single-agent case) — i.e. because there are periodic phases with no convergence. This periodic phenomenon is not explained for now.

3) Growing Density of Agents: Finally, we have conducted experiments with an increasing number of agents. They only concern EVAW+ — because other algorithms do not scale up — used on an 8x8 grid.

The results are presented on Table III. The main observation is that both the percentage of convergences to recurrent subgraphs and the time to convergence increase with the number of agents, but irregularly. These irregularities clearly coincide with settings where the grid size is not divisible by the number of agents, i.e. situations where the environment cannot be easily divided in equal length cycles (see Sec. II-D).



a- on an 8x8 grid



b- on a 14x14 grid

Figure 10. Convergence curves for one agent

One can notice that, on $(3, 8, +)$ and $(6, 8, +)$ settings, EVAW+ may not converge within the allowed time. This phenomenon is related to the recurrent graph detection algorithm. For the experiments presented in Table III, we limited the search depth to 3.000 nodes, hence restricting the size of detectable recurrent graphs to a maximum of 3.000 vertices. EVAW+ did actually converge to a recurrent graph but the detection algorithm was not able to find it. We also ran experiments on the same settings with larger search depths (up to 2.000.000 nodes). We observed a slight decrease of the probability of non-convergence with the search depth growth not yet leading to the disapearence of the phenomenon. As an example, with a 500.000 nodes depth the non-convergence rate dropped around 3.6% for the $(3, 8+)$ setting. As EVAW did not produce such results on the same settings ($(3, 8, -)$ see Table II), we can assume that the EVAW+ heuristic is likely to produce very large recurrent graphs (more than 2.000.000 nodes) under certain settings.

4) Decrease of the Number of Hamiltonian Cycles: Finally, one can also observe that the probability to converge to Hamiltonian cycles decreases when the grid size increases. It is not clear whether this reflects a property

setting	# runs	Ham. cycles	$length$ (Ham.)	non-H. cycles	$\overline{length}$ (non-H.)	absorbing subgraphs	no convergence	$\overline{time}$ $\overline{to\ conv.}$	$stdev(time$ $to\ conv.)$
1,8,d	20000	18.53 %	64	81.47 %	68	.00 %	.00 %	615	428
1,8,-	10000	26.79 %	64	73.21 %	66	.00 %	.00 %	596	401
1,8,+	10000	61.00 %	64	39.00 %	66	.00 %	.00 %	350	148
3,8,d	5000	.00 %	0	100.00 %	40	.00 %	.00 %	31283	30971
3,8,-	10000	.00 %	0	72.83 %	31	27.17 %	.00 %	71063	80072
3,8,+	10000	.00 %	0	80.83 %	84	13.31 %	5.86 %	3590	29000

Table I
SUMMARY OF EXPERIMENTS ON THE *determinization* OF EVAW.

setting	# runs	Ham. cycles	$length$ (Ham.)	non-H. cycles	$\overline{length}$ (non-H.)	absorbing subgraphs	no convergence	$\overline{time}$ $\overline{to\ conv.}$	$stdev(time$ $to\ conv.)$
1,8,-	10000	26.79 %	64	73.21 %	66	.00 %	.00 %	596	401
1,8,+	10000	61.00 %	64	39.00 %	66	.00 %	.00 %	350	148
1,14,-	10000	10.06 %	196	89.94 %	211	.00 %	.00 %	116200	115605
1,14,+	10000	15.82 %	196	84.18 %	199	.00 %	.00 %	2264	1485
2,8,-	10000	34.08 %	32	53.16 %	56	12.76 %	.00 %	4356	5728
2,8,+	10000	24.81 %	32	61.24 %	68	13.95 %	.00 %	644	824
2,14,-	2800	1.78 %	98	69.57 %	186	2.25 %	26.39 %	1194667	884023
2,14,+	9992	5.41 %	98	88.03 %	197	6.05 %	.50 %	7219	70516
3,8,-	10000	.00 %	0	72.83 %	31	27.17 %	.00 %	71063	80072
3,8,+	10000	.00 %	0	80.83 %	84	13.31 %	5.86 %	3590	29000
3,14,-	1300	.00 %	0	2.38 %	184	.23 %	97.38 %	1869325	851906
3,14,+	9991	.00 %	0	90.64 %	186	6.44 %	2.91 %	7558	23258

Table II
SUMMARY OF EXPERIMENTS CONDUCTED WITH EVAW AND EVAW+.

setting	# runs	Ham. cycles	$length$ (Ham.)	non-H. cycles	$\overline{length}$ (non-H.)	absorbing subgraphs	no convergence	$\overline{time}$ $\overline{to\ conv.}$	$stdev(time$ $to\ conv.)$
1,8,+	10000	61.00 %	64	39.00 %	66	.00 %	.00 %	350	148
2,8,+	10000	24.81 %	32	61.24 %	68	13.95 %	.00 %	644	824
3,8,+	10000	.00 %	0	80.83 %	84	13.31 %	5.86 %	3590	29000
4,8,+	10000	13.27 %	16	38.99 %	43	47.74 %	.00 %	4204	4924
5,8,+	3100	.00 %	0	58.77 %	95	41.22 %	.00 %	252878	278600
6,8,+	450	.00 %	0	42.88 %	46	52.88 %	4.22 %	561121	485924
8,8,+	3450	9.73 %	8	2.11 %	19	88.14 %	.00 %	88195	85185
10,8,+	125	.00 %	0	.00 %	0	4.80 %	95.20 %	993875	517571

Table III
SUMMARY OF EXPERIMENTS CONDUCTED WITH EVAW+ ON AN 8X8 GRID AND AN INCREASING NUMBER OF AGENTS.

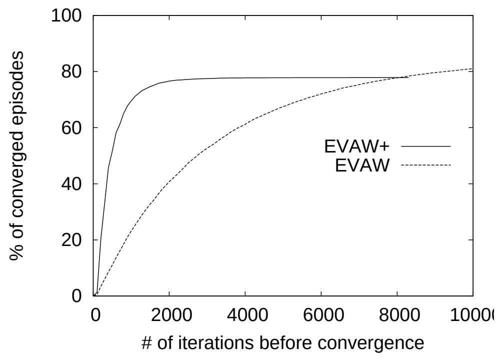
of the environment — e.g. that there are less Hamiltonian cycles among possible recurrent subgraphs — or a property of EVAW — its exploration behavior is less likely to find Hamiltonian cycles.

VII. DISCUSSION / CONCLUSION

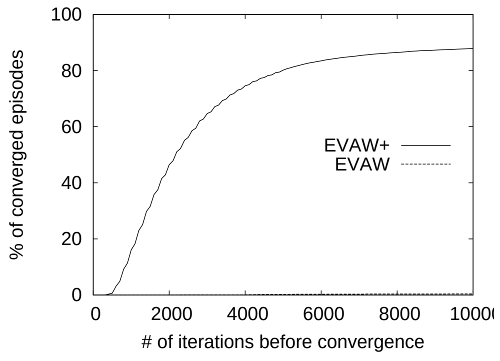
This paper studied the convergence behavior of a typical ant-based patrolling algorithm. Both Theorems 3.1 and 3.2 show that the system self-organizes: agents follow repetitive patterns which are guaranteed to be stable in the single agent case or if agents are deterministic. These results can be generalized to the other ant-based algorithms we mentioned, except Node Counting (because no upper-bound of its covering time is known).

Moreover, let us recall that all theoretical results or algorithms in this paper are not restrained to a particular graph topology, although most examples and experiments make use of grid environments. Other regular tilings — triangular or hexagonal — could be considered, but also irregular and even changing topologies — as in a website whose pages and relative links evolve with time.

We have also extended the Tortoise and the Hare algorithm to detect cycles and recurrent graphs in systems with non-deterministic dynamics. Brent's cycle detection algorithm [10] could be extended in the exact same way. Although distinguishing transient subgraphs from recurrent ones can be easily done — i.e. by constructing all possible futures from a given point of the simulation — it may be costly both in time and memory. Our approach here is to progressively build a partial graph representing the sequence of *system states* followed by the system and to regularly check whether the system has converged to a cycle or a

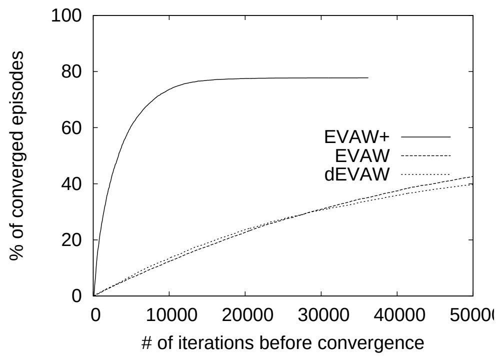


a- on an 8x8 grid

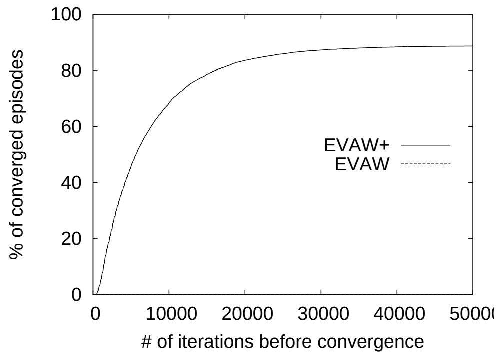


b- on a 14x14 grid

Figure 11. Convergence curves for two agents



a- on an 8x8 grid



b- on a 14x14 grid

Figure 12. Convergence curves for three agents

recurrent subgraph.

A third contribution is the proposal of the EVAW+ algorithm, which proves to converge to cycles much faster than EVAW, and more often to Hamiltonian cycles when they exist. When an Hamiltonian cycle is reached, this is an optimal solution in terms of visit frequency (idleness). Moreover, we observe that the other non-Hamiltonian cycles are often near-optimal (the length is usually close to a multiple of the number of cells). This means that the performance in terms of idleness is very good once convergence is attained.

More experiments need to be conducted to further evaluate EVAW and EVAW+, including during the exploration phase. In fact, finding or designing the "best" ant-based patrolling algorithm requires studying the influence of a variety of parameters: the value (or marking) update rule, the ordering of actions, the synchronization between agents, the determinization of conflicts and hesitations... It would be also interesting to see whether they are competitive with other approaches in both offline and on-line settings.

Appendix

A. Markov Chains

1) Introduction: A *Markov Chain* is a discrete time *Markov Process*. It is defined as a sequence $X_0, X_1, X_2, \dots$ of random variables over a state space S, this sequence verifying the *Markov Property*:

$$Pr(X_n | X_0, X_1, \dots, X_{n-1}) = Pr(X_n | X_{n-1}).$$

We only consider *homogeneous* Markov chains, i.e. chains where the transition mechanism is stationary:

$$Pr(X_n | X_{n-1}) = Pr(X_1 | X_0).$$

When S is finite, the Markov chain can be viewed (see Fig. 13) as a directed graph whose nodes are the states in S and where an edge $s \rightarrow s'$ represents a possible transition from s to s' and is valued with the probability $Pr(X_n = s' | X_{n-1} = s)$.

2) Various Types of Subgraphs: In the present paper —where only finite homogeneous Markov chains are encountered— we use the following notions:

- Two states $s, s' \in E$ *communicate* if and only if 1) $\exists\ n > 0$ such that $Pr(X_n = s' | X_0 = s) > 0$ and 2) $\exists\ n' > 0$ such that $Pr(X'_n = s | X_0 = s') > 0$. This defines an equivalence relation in which an equivalence class gathers all the states that communicate with each other.
- A *transient subgraph* is an equivalence class $E' \subset E$ such that, for all $s, s' \in E'$, there is a positive probability (but not certainty) to get to state s' from

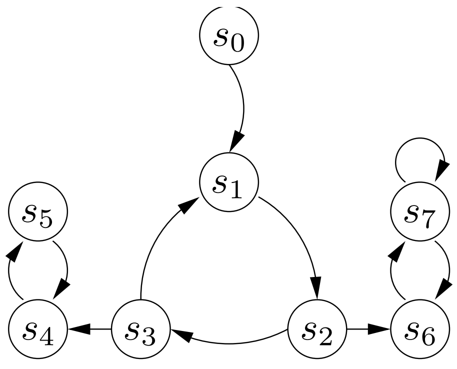


Figure 13. Finite homogeneous Markov chain represented as a Directed Graph (here without the transition probabilities)

state s, what can be formally stated as:

$$0 < \sum_{n=1}^{\infty} Pr(X_n = s' | X_0 = s) < 1.$$

This means that the system considered can "loop" in this subgraph for some time but that at some point it will leave this subspace forever.

- A *recurrent subgraph* is an equivalence class $E' \subseteq E$ such that, for all $s, s' \in E'$, $\sum_{s' \in E \setminus E'} Pr(s'|s) = 0$. There is no possible transition from a state s in E' to a state outside of E'.
- A recurrent subgraph in which all transitions are deterministic ($Pr(s'|s) \in \{0, 1\}$) is called a *cycle*.

On Figure 13 we have:

- 3 equivalence classes: $\{s_1, s_2, s_3\}$, $\{s_4, s_5\}$ and $\{s_6, s_7\}$;
- 1 transient subgraph: $\{s_1, s_2, s_3\}$;
- 2 recurrent subgraphs: $\{s_4, s_5\}$ and $\{s_6, s_7\}$; and
- 1 cycle: $\{s_4, s_5\}$.

Note that the visual representation can be simplified by removing most of the nodes followed by deterministic transitions (i.e. with a single outgoing edge).

References

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

[2] 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)*, 2008.

[3] 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.

[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] 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)*. Springer-Verlag, 2004, pp. 474–483.

[6] S. Koenig, B. Szymanski, and Y. Liu, "Efficient and inefficient ant coverage methods," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1–4, 2001.

[7] 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)*, 2007, pp. 442–449.

[8] S. Thrun, "Efficient exploration in reinforcement learning," Carnegie Mellon University, Tech. Rep. CMU-CS-92-102, 1992.

[9] R. Floyd, "Non-deterministic algorithms," *Journal of the ACM*, vol. 14, no. 4, pp. 636–644, October 1967.

[10] R. Brent, "An improved Monte Carlo factorization algorithm," *BIT*, vol. 20, pp. 176–184, 1980.

[11] D. E. Knuth, *The Art of Computer Programming, vol. II: Seminumerical Algorithms*. Addison-Wesley, 1969.

[12] F. Michel, G. Beurier, and J. Ferber, "The TurtleKit simulation platform: Application to complex systems," in *Proceedings of the International Conference on Signal-Image Technology and Internet-Based Systems (SITIS'05)*, 2005.