Termination and Stability Levels in Evolved CA Agents for the Black–Pattern Task

Rolf Hoffmann[1(✉)], Dominique Désérable[2], and Franciszek Seredyński[3]

[1] Technische Universität Darmstadt, Darmstadt, Germany
hoffmann@informatik.tu-darmstadt.de
[2] Institut National des Sciences Appliquées, Rennes, France
domidese@gmail.com
[3] Department of Mathematics and Natural Sciences,
Cardinal Stefan Wyszynski University, Warsaw, Poland
fseredynski@gmail.com

Abstract. Given a $2d$ Cellular Automaton (CA) with mobile agents controlled by a finite state automaton (algorithm). Initially the field is colored white and agents are randomly placed. They have the task to color the whole field into black in shortest time. The objective is to find algorithms that (1) can form the *black*–pattern, (2) keep it *stable* and then (3) change into a global state where all agents *stop* their activity. Four levels of stability are distinguished, depending on the grade of inactivity after having formed the pattern. For systems with up to four agents we found such algorithms by applying genetic algorithms (GA) and manual post fine tuning. Performances and simulations of these algorithms are presented.

Keywords: Termination in multi-agent systems · Stability
Cellular automata agents · Pattern formation · Genetic algorithm
Spatial computing

1 Introduction

The Problem. Initially all $N = n \times n$ cells of a square field with border are colored white, k agents are there randomly placed and their direction is also random. The CA multi-agent system ("CA–MAS") has to solve the *Black–Pattern* task with *Termination*, shortly the "BPT". That is, the agents must explore and *color* the whole cell field from white into black in shortest time and keep it stable, and then they must *stop* moving around and turning. This means that not only a stable output is required, but also a kind of termination for the entire multi-agent system. The agents shall be controlled by a finite state automaton with a minimal number of states. Actions and inputs must be very limited and local. Although this task sounds easy to accomplish, that is not the case, especially with more than one agent.

G. Mauri et al. (Eds.): ACRI 2018, LNCS 11115, pp. 137–145, 2018.
https://doi.org/10.1007/978-3-319-99813-8_12

The underlying general objective is to study the termination in CA–MAS and thereby to motivate further research thereon. For this purpose, and in order to keep the complexity as low as possible, we revisit an already studied very simple CA, the *Creature's Exploration Problem* [1], except that neither color (for indirect communication) nor termination were considered therein. Incidentally, the *basic* action of *blacking* a cell may also be interpreted as a *control* message – a primitive signal, a marker, a trace, a *stigma*. According to [2], *stigmergy* is "the process of indirect communication of behavioral messages with implicit signals" and where *indirect* means the "interaction through the environment". Thus, the cell color acts also as a very limited distributed communication memory. It is worth to emphasize the difference of nature between a *static* trace deposited in a cell and a *dynamic* message traveling through a channel in distributed computing. Stigmergy is now a feature widely highlighted in many environments [3].

Related Work to Termination. Termination detection is a fundamental problem in distributed computing. A set of processes execute a task and communicate through interprocess channels by messages. The computation is entering a quiescent state as soon as all processes are idle and all channels are empty. Since there is neither global clock nor a common memory, the detection of a global quiescence is impossible without an additional *control* mechanism which should not interfere with the *basic* computation. The pioneering works of Lamport, Dijkstra–Scholten, Francez, Misra–Chandy are well known and a lot of others in the eighties thereon. All those control schemes are categorized, at least until 1998, in an elegant taxonomy including eight classes [4]. Termination detection is also a fundamental problem in multi-agent "MAS" systems. As a matter of fact, there is a close relationship between distributed systems and multi-agent systems, although some dissimilarities can be highlighted [5,6]. We consider their differences as minor and thereby that MAS termination detection procedures could enter Matocha–Camp taxonomy [4], at least updated.

In Sect. 2, the termination problem is defined through the black–pattern task and four stability levels are proposed. In Sect. 3, the FSM–based multi–agent system is presented. Then k–agent algorithms are analyzed in Sect. 4 with $k = 1, 2, 4$ and various scenarios of stability and termination are studied as well as performances and robustness before Conclusion.

2 Termination and Stability Levels

The problem of termination in CA–MAS was already noticed in [7]. How can a multi-agent system be stopped in a decentralized way after having formed the required pattern? Like in [4], a simple way would be to flood the CA network with a *wave* at each time-step. This technique requires a lot of additional resources and is very time-consuming. Therefore we were looking for a more effective way. The idea is, that during the run (without a separate wave phase after each time-step), the agents themselves are able to detect that the pattern was formed (or will safely be formed in the near future) and then automatically stop their activities.

Thereby the energy consumption of the whole system stops or is minimized after the job is done. Another advantage is the following: when the agents recognize that the task is accomplished, they are able to trigger a new task. This is an important feature allowing to execute a sequence of subtasks in a decentralized way. Time is counted in discrete time steps $t = 0, 1, ...$, because CA agents are working in the synchronous CA model. We define four levels $\Lambda_0 \prec \Lambda_1 \prec \Lambda_2 \prec \Lambda_3$ of stability. The precedence means that Λ_j is stronger than Λ_i for $j > i$.

- Λ_0 – *Unstable*: the aimed pattern is formed for the first time at $t = T_0$. After that, the pattern may change.
- Λ_1 – *Stable*: the aimed pattern is formed and remains stable for time $t \geq T_1$ and at least one agent continues moving around.
- Λ_2 – *Stable idle-stop*: the aimed pattern is formed and remains stable for time $t \geq T_2$, all agents have stopped moving around at time $T_2^{stop} \geq T_2$ and at least one agent continues turning. We call such algorithm *idle–stopping*.
- Λ_3 – *Stable full-stop*: the aimed pattern is formed and remains stable for time $t \geq T_3$ and all agents have stopped moving around and turning at time $T_3^{stop} \geq T_3$. No activity is visible and we call such algorithm *full–stopping*.

Note that at level Λ_3 all agents become passive from the global observer's point of view. Nevertheless passive agents may change their internal state or may enter into in a special final dead-state. Note also that the agents need not to stop at the same time. If each agent wants to be informed about the termination, an additional *consensus* operation among the agents is necessary.

3 The Designed Multi-agent Cell Architecture

At first we have to design a cell architecture which is able to model agents, potentially can solve the problem, and is relatively simple in terms of "hardware" elements. It has to be tailored to a certain extent to the problem in order to solve it at all. Such an architecture consists in basic hardware elements, such as registers, memories, combinatorial logic and wires. We assume a synchronous

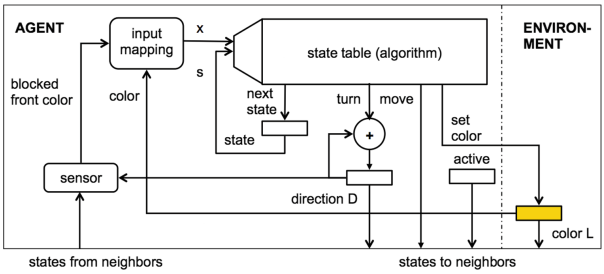

Fig. 1. The architecture of a cell. The state table defines the agent's next control state, its next direction, and whether to move or not. It defines also the setting of the color (0/1) as part of the environment.

working principle. One part of the architecture shall be fixed, and another part shall be configurable. The configurable part can be seen as a program, that allows to define the functionality within certain limits. For example a classical $1d$ CA cell consists of a 1–bit register (holding the state), a rule function (fixed, configurable or even variable logic or table), a feedback loop for the state, and wires in and between the cells. Such a classical cell corresponds to a simple Moore automaton.

Here we use a more complex cell, where the cell rule depends on the agent's current state (taking the history into account), and the rule is only executed on a site where an active agent is situated. The whole model is still fully compatible with the CA model, and therefore we use the term CA agent system. The designed cell architecture is depicted in Fig. 1. The whole *cell state* is stored in a composition of several registers:

$CellState = (Color, AgentState)$
  $Color\ L \in \{0, 1\}$
  $AgentState = (Active, Identifier, Direction, State)$
    $Active \in \{\texttt{true}, \texttt{false}\}$
    $Identifier\ ID \in \{0, 1, ..., k-1\}$
    $Direction\ D \in \{0, 1, 2, 3\} \equiv \{\texttt{toN}, \texttt{toE}, \texttt{toS}, \texttt{toW}\}$
    $State\ S \in \{0, 1, ..., N_s - 1\}$. // control state, initially set to zero

Each cell contains a color (as part of the environment) and one agent, which is either active and visible, or passive and not visible. When an agent is moving from cell A to cell B, *AgentState* is copied from A to B and the $Active$ bit of A is set to false. The first cell ahead (front cell) in the moving direction and the second cell ahead (in order to detect conflicts) are the neighbors.

An agent is controlled by a Mealy automaton, consisting of the state register s and the transition function, which here is defined by a state transition/output table, a *state table* for short. Table inputs are the control state s and defined input situations x, table outputs are the signals *nextstate*, *turn*, *move* and *setcolor*. The signal *nextstate* defines the next control state of the automaton. The *turn* signal triggers the change of the direction. The *move* signal is interpreted by the agent itself and is presented to the neighboring cells. The *setcolor* signal defines the setting of the color. The Mealy automaton realizes the "brain" or control unit of the agent. The state table can also be seen as a program or algorithm. Therefore we call the state table also *agent's algorithm* "AA". The state table corresponds to the genome (configurable part of the architecture) to be optimized by GA.

An agent has a moving direction D that also selects the cell in front as the actual neighbor. An agent can interpret the following conditions: *color*: cell color L, *front color*: front cell's color L_F, *blocked by border*: then the front color is defined as $L_F = -1$, *blocked by another agent*: either another agent is situated in front, or another agent with a higher priority wants to move to the same target cell in front. The *sensor* is responsible for the reduction of the neighboring states to the conditions *blocked* and *front color*, then further used by the input mapping.

Table 1. Input mapping function with $N_x = 10$ inputs.

	blocked	color	front color	x
blocked by border	1	0	-1	0
	1	1	-1	1
free	0	0	0	2
	0	0	1	3
	0	1	0	4
	0	1	1	5
blocked by agent	1	0	0	6
	1	0	1	7
	1	1	0	8
	1	1	1	9

Triggered by the state table output signals, the following actions are performed: **next state**: $state \leftarrow nextstate \in \{0, ..., N_s - 1\}$, **move**: $move \in \{0, 1\} \equiv \{wait, go\}$, **turn**: $turn \in \{0, 1, 2, 3\}$. The new direction is $D(t + 1) \leftarrow (D(t) + turn)\ mod\ 4$, **set color**: $setcolor \in \{0, 1\} \equiv \{color0, color1\}$. The new color is $L(t + 1) \leftarrow setcolor$.

All actions are performed in parallel. There is only one constraint: when the agent's action is *go* and the situation is *blocked*, then an agent cannot move and has to wait, but still it can turn and change the cell's color.

An input mapping function is used to limit the size of the state table memory. The *input mapping* reduces all possible input combinations to an index $x \in X = \{0, 1, \ldots, N_x - 1\}$ used in combination with the control state to select the actual line of the state table. The input mapping was defined as shown in Table 1.

Note that the hardware resources and capabilities (sensed situations, action set) are quite limited, which makes the given task with automatic termination difficult to solve. Moreover, the agents have not any knowledge about north–east–south–west orientation, that makes the task more complicated and more universal.

4 Multi-agent Algorithms

Algorithms for k-agent systems "k–AA" ($k = 1, 2, 4$) with different termination conditions were evolved by GA with manual improvement[1]. More details of the used GA method are given in [8]. Note that finite state algorithms were evolved, each represented as a state stable (the genome). The number of desired control states and the desired stability level were used as input parameters.

1–Agent Algorithm. A full-stopping algorithm was partly found by GA, and then manually improved. It needs only three states (Fig. 2). Zero is the initial

[1] The GA method was very time consuming (millions of multi-agent simulations) and took around 4 weeks of computation time on a state–of–the–art quad-core PC 3.5 GHz.

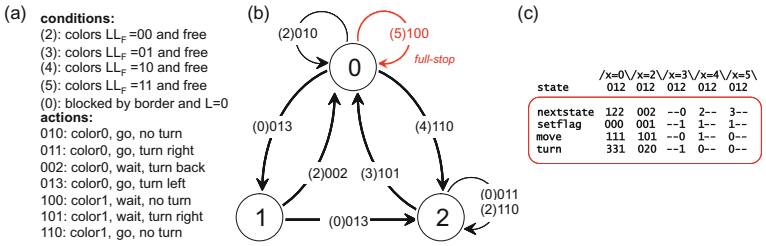

Fig. 2. The full-stopping 1-agent algorithm with 3 states. (a) Conditions and actions used in graph (b), (c) corresponding state table with don't cares (-).

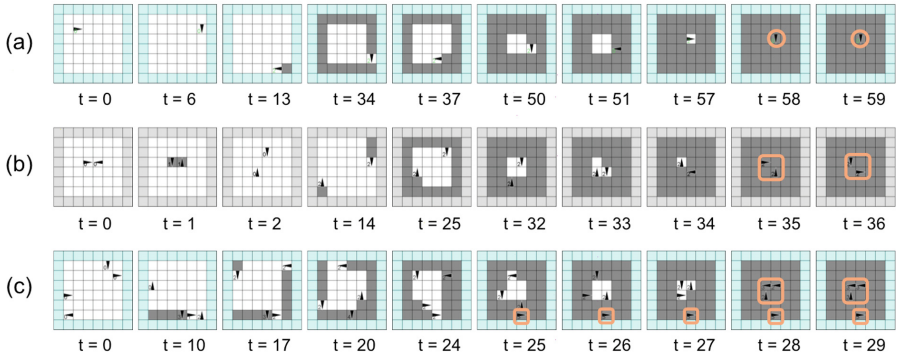

Fig. 3. (a) Simulation of the full-stopping 1–AA in a 6×6 field, $T_3^{stop} = 58$ (b) Simulation of the idle-stopping 2–AA, $T_2^{stop} = 35$ (c) Simulation of the full-stopping 4–AA, starting randomly, $T^{stop} = 28$.

and final state. The strategy can be understood by looking at a simulation (Fig. 3a). At first the agent searches for a corner, moving straight and turning right when detecting a border. After having found the corner, it starts to color the cells black, first moving along the borders and then moving inwards towards the center in a spiral-like trajectory. Then the agent stops moving and turning at $t \geq T_3^{stop}$. This full-stop corresponds to the self-loop in state 0 by Condition 5 (`colors` $LL_F = 11$) and Action 100 (`color1, wait, no turn`).

This full-stopping algorithm can easily be changed into an idle-stopping algorithm by changing the final actions from (`color1, wait, no turn`) into (`color1, wait, turn`). And it can be changed into a weaker algorithm (with stability level < 2) by changing actions into (`color1, go, turn/no turn`).

The number of needed time-steps is given in Table 2. Its time-complexity is linear in $O(N)$, an exact formula $t(n)$ could be derived by a simple analysis. The most time-consuming part is the coloring in a spiral-like way, in addition some steps are needed to detect borders, corners and already painted cells.

Table 2. Full-stopping 1–AA: number of time steps. Average is over 1000 fields.

Size	4×4	5×5	6×6	7×7	8×8	9×9	10×10
$T^{stop}_{3,mean}$	27.76	39.80	53.73	69.87	88.01	108.07	129.87
$T^{stop}_{3,min}$	24	35	48	63	80	99	120
$T^{stop}_{3,max}$	31	44	59	76	95	116	139
$T^{stop}_{3,mean}/N$	1.74	1.59	1.49	1.43	1.38	1.33	1.30

Table 3. Average time steps per cell T^{stop}/N. Full-stopping and idle-stopping 2–AA, evolved on 10×10 fields, simulated on 1000 fields for each field size.

	Size	4×4	5×5	6×6	7×7	8×8	9×9	10×10
Full-stop 2–Agent system	T^{stop}_3/N	1.22	1.09	1.01	0.96	0.72	0.90	0.87
Idle-stop 2–Agent system	T^{stop}_2/N	1.19	1.04	0.96	0.89	0.84	0.81	0.78
Full-and-idle-stop 1-A. S.	T^{stop}/N	1.74	1.60	1.49	1.43	1.38	1.33	1.30

2–Agent Algorithm. The 2-agent algorithm was first evolved by GA on 6×6 fields. It turned out that the found algorithms were not working well on other field sizes. Therefore GA optimization was performed on one thousand 10×10 training fields with 4 states. Then the found idle-stopping algorithm was manually changed into a full-stopping algorithm.

A simulation sample is shown in Fig. 3b. At first, the agents search for corners. Then they start paint black in spiral-like way, with two active opposite coloring points. The agents in the idle-stopping algorithm continue turning after coloring whereas they fully stop in the full-stopping algorithm.

The performance for different field sizes is shown in Table 3. The 2-agent algorithm executed on 1-agent systems yields the same performance as the former 1–AA in Table 2. A full-stopping 10×10 system with 2 agents is 1.49 times faster (1.30/0.87) than a system with 1 agent only. And a full-stopping 10×10 system with 2 agents is 1.12 times slower (0.87/0.78) than an idle-stopping system. In order to compute T^{stop} average values, 1,000 random fields were simulated.

Table 4. Performance of 4–AA on 6×6 fields. Full-stopping (*left*), idle-stopping (*right*). Values are averaged over 1000 fields.

	T^{stop}	T^{stop}/N	cost per cell $k\,T^{stop}/N$	relative speedup
4-agent system	30.46	0.85	3.38	2.07
2-agent system	43.95	1.22	2.44	1.50
1-agent system	58.73	1.63	1.63	1.00

	T^{stop}	T^{stop}/N	cost per cell $k\,T^{stop}/N$	relative speedup
4-agent system	26.53	0.74	2.95	1.87
2-agent system	38.96	1.08	2.16	1.37
1-agent system	56.73	1.58	1.58	1.00

4–Agent Algorithm. Many computer-time consuming attempts were made by GA to find a 4-agent algorithm with 6 states that can work successfully on any field size. Until now, no general algorithms were found. Nevertheless GA was able to find specialized algorithms that work on one thousand random fields of size 6×6 or 10×10.

If the agents start rotational symmetrically then they are first searching for corners and then they build the pattern using a counter-clock spiral trajectory. When the agents start randomly (Fig. 3c) then the pattern building is slower and not so symmetric, but still the tendency of building a counter-clock spiral can be observed.

The full-stopping 4-agent system is $30.46/26.53 = 1.15$ times slower than the idle-stopping system (Table 4). The cost per cell is the number k of agents multiplied with the number of needed time units per cell. So the 4-agent systems are about twice more costly than the 1-agent systems, while they are about two times faster.

5 Conclusion

In this paper four stability levels for the termination of CA multi-agent system were proposed. *Idle-stopping* and *full-stopping* algorithms were found for the BPT where the whole field has to be painted from white to black in shortest time. The general 1-AA need only 3 states and are relatively fast with time-complexity $O(N)$. The general full-stopping 2–AA needs 4 states and is about 50% faster than the 1–AA. Until now, no general 4–AA was found, but special ones for fields of size 6×6 and 10×10 were evolved by GA. They work about twice as fast as 1–AA. All found algorithms follow in principle the same strategy: first searching for corners, then follow a spiral-like trajectory until the midpoints are reached. Future work is directed to find general algorithms that can work successfully on any field size with any number of agents. Another topic is the efficient communication and synchronization of the stopping state between agents.

References

1. Halbach, M., Hoffmann, R., Both, L.: Optimal 6-state algorithms for the behavior of several moving creatures. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 571–581. Springer, Heidelberg (2006). https://doi.org/10.1007/11861201_66
2. Tummolini, L., Castelfranchi, C.: Trace signals: the meanings of stigmergy. In: Weyns, D., Parunak, H.V.D., Michel, F. (eds.) E4MAS 2006. LNCS (LNAI), vol. 4389, pp. 141–156. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71103-2_8
3. Weyns, D., Van Dyke Parunak, H., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems State-of-the-art and research challenges. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS (LNAI), vol. 3374, pp. 1–47. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-32259-7_1

4. Matocha, J., Camp, T.: A taxonomy of distributed termination detection algorithms. J. Syst. Softw. **43**(3), 207–221 (1998)
5. Wellman, M.P., Walsh, E.W.: Distributed quiescence detection in multiagent negotiation. In: Fourth International Conference on Multi-Agent Systems, ICMAS, pp. 317–324 (2000)
6. Lahlouhi, A.: MAS-td: an approach to termination detection of multi-agent systems. In: Gelbukh, A., Espinoza, F.C., Galicia-Haro, S.N. (eds.) MICAI 2014. LNCS (LNAI), vol. 8856, pp. 472–482. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13647-9_42
7. Hoffmann, R., Désérable, D.: Generating maximal domino patterns by cellular automata agents. In: Malyshkin, V. (ed.) PaCT 2017. LNCS, vol. 10421, pp. 18–31. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62932-2_2
8. Hoffmann, R.: How agents can form a specific pattern. In: Wąs, J., Sirakoulis, G.C., Bandini, S. (eds.) ACRI 2014. LNCS, vol. 8751, pp. 660–669. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11520-7_70