How Agents Can Form a Specific Pattern

Rolf Hoffmann

Technische Universität Darmstadt,
FB Informatik, FG Rechnerarchitektur,
Hochschulstr. 10, 64289 Darmstadt, Germany
hoffmann@informatik.tu-darmstadt.de

Abstract. A multi-agent system is considered, comprised of a square 2D cell field of cells with uniform agents controlled by finite state machines (FSMs). Each cell contains a particle with one out of four colors, which can be changed by the agents. Initially the agents and colors are randomly distributed. The objective is to form a specific target pattern belonging to a predefined pattern class. The target patterns (path patterns) shall consist of preferably long narrow paths with the same color. The quality of the path patterns is measured by a degree of order, which is computed by counting matching 3 x 3 patterns (templates). The used agents can perform 32 actions, combinations of moving, turning and coloring. They react on the own color, the color in front, and blocking situations. The agents' behavior is determined by an embedded FSM with 6 states. For a given 8 x 8 field, near optimal FSMs were evolved by a genetic procedure separately for $k = 1$ .. 48 agents. The evolved agents are capable to form path patterns with a high degree of order. Agents, evolved for a 8 x 8 field, are able to structure a 16 x 16 field successfully, too. The whole multi-agent system was modeled by cellular automata. In the implementation of the system, the CA-w model (cellular automata with write access) was used in order to reduce the implementation effort and speed up the simulation.

Keywords: Multi-Agent System, Cellular Automata Agents, Pattern Formation, Evolving and Learning FSM Behavior, CA-w.

1 Introduction

The Agents'Task. Given is a square field of $N = n \times n$ cells with border, and we assume an even number for n. Each cell, except the border cells, contains a particle with a certain *color* $\in \{0, 1, 2, 3\}$. A given number k of agents can move around in the field and can change the colors at the sites they are situated on. Initially the colors, the agents, and the agent's directions are randomly distributed. The task is to end up in a global state where a certain *target pattern* appears, belonging to a predefined pattern class. As an example we defined the *path pattern class* which is characterized by preferably long paths of width = 1, where all the neighboring cells have another color or are border cells. The paths may have branches and may form loops (as shown in Fig. 5b). The objective is to find the behavior of the agents that can solve this task with a certain quality. The capabilities of the agents shall be constrained, e.g. the number of control states, the action set, and the details of the perceived environment.

Motivation. The original idea for this research was to find artificial patterns which are to a certain extend creative and impressive from the artistic point of view. As artistic patterns are difficult to evaluate in a formal way, the more modest objective was to find patterns with a certain interesting or valuable structure. Then experiments were conducted to find certain global patterns, like a black box in the center of the field, or with a global symmetry. It turned out, until now, that it is very difficult to find agents that can comply with such strict global objectives. Then, the objective was even more relaxed, namely to find global patterns that obey to intrinsic local rules (local matching patterns, templates). For this work, the Moore-Neighborhood (3 x 3) was used for the templates, but it could be enlarged in order to form more interesting global patterns. – Benefit of the presented results could also be taken when nano-structures have to be constructed by nano-robots or by beaming focused energy onto certain cells in order to change their physical state [1–3]. Other applications of this research can be imagined when the task is to form biological [14], chemical or computational structures with specific properties.

Why Agents? What is the advantage to solve this task by agents? Generally speaking, agents can behave very flexible, powerful and coordinated because of their intelligence and their specific sensors and actuators. Important properties that can be achieved by agents are

– *Scalability.* The problem can be solved with a variable number of agents, and faster or better with more agents.
– *Tuneability.* Increasing the agent's intelligence, the problem can be solved faster or more effective (better quality of solutions).
– *Versatility.* Similar problems can be solved by the same agents, e.g. by changing the shape or size of the environment.
– *Updating-tolerance.* Usually the time-evoluted global state depends only marginally on the updating-scheme (synchronous, asynchronous).
– *Fault-tolerance.* When obstacles are introduced or not all agents work correctly, the problem can still be solved in a gracefully degraded way.

Because agents are very flexible, they can be employed to design, model, analyze, simulate, and solve problems in the areas of complex systems, real and artificial worlds, games, distributed algorithms and mathematical questions.

Related Work. (i) *FSM controlled agents:* In former investigations we have tried to find the best algorithms for the *Creature's Exploration Problem* [4], in which the agents have the task to visit all empty cells in shortest time, for the *All-to-All Communication Task* [5], in which each agent has to distribute its information to all the others, and for the *Target Searching Task* [7]. The FSMs for these tasks have been evolved using, i. e., genetic algorithms, genetic programming [6], and sophisticated enumeration methods. Other related works are a multi-agent system modeled in CA for image processing [8], and modeling the agent's behavior by an FSM with a restricted number of states [9]. An important pioneering work about FSM controlled agents is [10].

(ii) *Pattern-formation:* Agent-based pattern formations in nature and physics are studied in [15, 16]. A programming language is presented in [17] for pattern-formation of locally-interacting, identically-programmed agents; as example the layout of an CMOS inverter is formed by agents. In [19] a general framework is proposed to discover rules that produce special spatial patterns based on a combination of Machine Learning strategies including Genetic Algorithms and Artificial Neural Networks. In [20] a methodology based on different learning-techniques is introduced that helps the designer to model the behavior of the agents for a multi-agent system.

(iii) *Modeling moving agents/particles:* Here the agents are modeled by classical CA or CA-w as described in Sect. 3. Other modeling concepts related to CA are lattice-gas cellular automata, block substitutions, or partitioned CA as used in [5]. The concept of transactional CA was proposed in [18] to implement agent mobility in CA, and it was shown that scheduling policies and conflict resolution strategies have an impact on the global behavior of the system.

2 Target Patterns and Degree of Order

How can the class of target patterns be characterized? The idea is to use a set of small local matching patterns as building blocks, also called *templates*, that can successfully tile the field (with overlaps). In the color structures arranged by the agents such templates are expected with a high frequency. The 41 templates used here are shown in Fig. 1a. They describe the target *path patterns*. E.g. the *plank* template means that there are 3 consecutive cells of the same color (depicted in black), enclosed by 3 + 3 cells in another color (depicted in grey). The plank can be rotated by 90 degrees, giving the second form of this type. Altogether, under reflection and rotation, there are 41 distinct templates. The templates can be depicted in a condensed form (Fig. 1b), using *don't care* neighbors that can have any color. E.g. the template T4 does not care about the West and East neighbor.

The patterns created by the agents have to be evaluated, how well do they fit into the defined path pattern class. In order to evaluate a given pattern, all templates are applied (tested) on each cell. If a match (hit) is found, a dot is used to mark this cell. Then, all dots are summed up which gives the total number of hits h. This number is also called *degree of order*. The terminal cells of a path will not be taken into account when the path length is computed. Thus a consecutive path of 3 cells has only a length count of 1. The reason is to avoid searching for very short paths of length ≤ 2. The theoretically maximum order is $h_{max} = n \times n - 4$. The relative order is h/h_{max}. The maximum could be reached by nesting squares with different colors into each other. In Fig. 1b there are 6 templates that match, leading to 9 hits.

It can be observed that the testing for templates hits corresponds to the application of an equivalent CA rule. From the formal language point of view, the templates can be seen as the generators of a 2D pattern language.

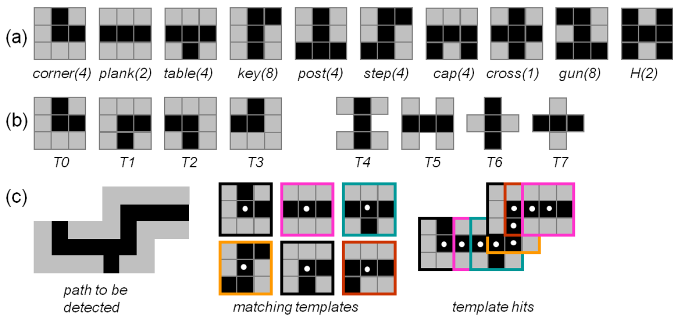


Fig. 1. (a) Templates are small building blocks which are expected to appear in a target pattern. A path cell is colored in black. Grey represents another color. The number of symmetric templates by rotation and reflection is given in the brackets. (b) The same templates, described in condensed form. (c) A path (part of the target pattern) can be tiled (with overlaps) by matching templates. Each matching template produces a hit (dot). All hits are summed up and give the degree of order.

3 Modeling the System by Cellular Automata

Standard in CA is that the cells are uniform, meaning that they are all similar and obey to the same rule. The rule changes the state of each cell by taking into account the own cell's state and the states of its neighbors. Nevertheless the cell's rule has to react on non-uniform situations, e.g. whether there is an agent situated on a cell or not. Therefore the cell's state is modeled as record $(Type, Color, Agent)$ comprising a type tag, where

$Type \in \{Border, Particle, AgentAndParticle\}$, and

$Agent = (Identifier, Direction, ControlState)$.

When designing a system with agents (multi-agent system MAS), then the capabilities of the agents have to be defined before designing or searching for the behavior of the agents to solve a given task. The main capabilities are: The perceivable inputs from the environment, the actions an agent can perform, and the size of its memory (number of possible control states, optionally additional data states). In our system, an agent shall react on the following inputs in a certain combination

- the **own color** C of the cell the agent is situated on
- the **color in front** C_F (in moving/viewing direction)
- a **border** cell in front
- the **blocked** situation/condition, caused either by a border, another agent in front, or when another prior agent can move to the front cell in case of a conflict. The inverse condition is called *free*.

An agent has a moving/viewing $Direction = D \in \{0, 1, 2, 3\}$ =\{*toN, toE, toS, toW*\}. Note that in the used model an agent cannot observe the direction and control state of another agent in the neighborhood. The actions that an agent shall be able to perform are

- **move**: $move \in \{0, 1\} = \{wait, go\}$

– **turn**: $turn \in \{0, 1, 2, 3\}$. The new direction is $D(t+1) = (D(t)+turn)\ mod\ 4$.
– **set color**: $setcolor \in \{0, 1, 2, 3\}$. The new color is $C(t+1) = setcolor$.

The move, turn and set color actions can be performed simultaneously (32 combinations). There is only one constraint: when the agent's action is *go* and the situation is *blocked*, then the agent cannot move and has to wait, but still it can turn and change the color. In case of a moving conflict, the agent with the lowest identifier (ID $= 0 \,..\, k-1$) gets priority. Instead of using the identifier for prioritization, it would be possible to use other schemes, e.g. random priority, or a cyclic priority with a fixed or space dependent base.

How can an agent move from A to B in the cellular automata (CA) model? Two rules have to be performed, a *delete-rule* that deletes the agent on A, and a *copy-rule* that copies the agent to B. In CA both rules have to compute the same blocking condition, this means a redundant computation. In order to avoid this redundancy, a two-phase updating scheme could be used (first compute the moving condition, second use it in cell A and B), or the *cellular automata with write-access model* (CA-w) [11]. When using the CA-w model, the moving condition is computed by cell A, and if it is true, A applies a rule that deletes the agents on A and copies it to B. The simulation program was implemented by the CA-w model, although it is possible to implement the system in standard CA with redundant computation.

The CA-w model was introduced in order to describe moving agents, moving particles or dynamic changing activities. This model allows to write information onto a neighbor. This method has the advantage that a neighbor can directly be activated or deactivated, or data can be sent actively to it by an agent. The CA-w model is a restricted case of the more general, "*Global*" GCA-w model [12, 13]. In GCA-w any cell of the whole array can be modified whereas in the CA-w model only the local neighbors can be. Usually the cells of these models are a composition of (data, pointers). The neighbors are accessed via pointers, that can be changed dynamically like the data by an appropriate rule from generation to generation. Comparing CA and CA-w, a CA equivalent to a CA-w with neighborhood N_1 can be found by extending N_1 to N_2 (N_1 extended by write-distance). For example, an 1D CA–w with neighborhood distance 1 (read and write) is equivalent to a CA with neighborhood distance 2.

The behavior of an agent shall be determined by an embedded finite state control automaton (FSM) (Fig. 2). We also formulate that an agent has/obeys to a certain (control) algorithm. Each CA cell contains an FSM which is active when an agent is situated on it. The FSM contains a *state table* (also called *next state/output table*). Outputs are the actions (move, turn, setcolor) and the next control state. Inputs are the control state and the relevant input situations x. The *input mapping* reduces all possible input combinations of (border, blocked, color, front color) to an index $x \in X = \{0, 1, \ldots, |X| - 1\}$ that is used in combination with the control state to select the actual line of the state table.

The following input mapping is used. If the situation is *free* the index x is the color in front: $x = C_F$. If the situation is *blocked* by a border cell in front, then $x = 4$. If the blocking is caused by another agent in front, or by a prior agent in the case of a conflict, then the own color is directly mapped to the index with an offset $x = C + 5$. It is possible to choose other input mappings, with less or more x values, or other assignments, e.g. in the case of blocking, the direction of the agent could be used ($x = D + 5$), instead of the own color.

The used updating scheme is *synchronous*; exemplarily simulation experiments showed, that the results with asynchronous updating are quite similar.

4 Evolving the Agent's Behavior by a Genetic Procedure

The goal is to find a general applicable FSM which is optimal for a large set of different initial configurations covering the whole area of applications (different size and shape of the field, different number of agents). As we cannot optimize for a very large set of initial configurations within a limited amount of computation time, we used a fixed field size of 8 x 8 and optimized separately for $k = 1, 2, 4, 8, 16, 32, 48$ agents, and used 100 training fields for each k. This means that we searched for specialists and not for all-rounders.

As the search space for different FSMs (algorithms) is very large, we are not able to check all possible behaviors by enumeration. The number of FSMs which can be coded by a state table is $Z = (|s||y|)^{(|s||x|)}$ where $|s|$ is the number of control states, $|x|$ is the number of different inputs and $|y|$ is the number of different outputs. As the search space increases exponentially, we use a genetic procedure in order to find the best behavior with reasonable computational cost. Even with a genetic approach the number of states, inputs and outputs have to be kept low in order to find a good solution in acceptable time.

A possible solution (genome of one individual in the genetic) corresponds to the contents of the FSM's state table (Fig. 2b). The column index j is defined by a certain combination of *(x, s)*. Each column j defines *(s', y) = (nextstate, setcolor, move, turn)*. The used genetic procedure per iteration is

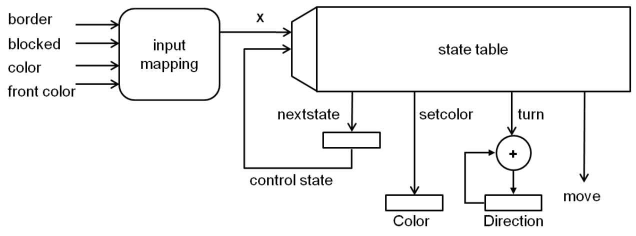
1. $A' \leftarrow mutate(A)$
2. $(A, B) \leftarrow deleteDuplicates(sort(A, A', B))$
3. $(A, B) \leftarrow exchange(b, A, B)$.

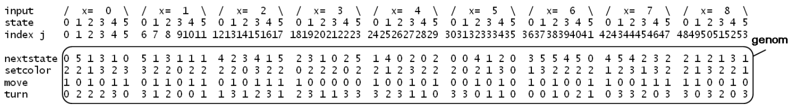
One population of N individuals is stored in two lists (A, B) with $N/2$ individuals each. (Step 1) During each iteration $N/2$ offspring are produced from list A by mutation. (Step 2) The union of the current N individuals and the $N/2$ offspring are sorted according to their fitness, duplicates are deleted and the number of individuals is then reduced to the limit of N in the pool. (Step 3) In order not to get stuck in local minima and to allow a certain diversity in the gene pool, the first b individuals from B are exchanged with the last b individuals from A. We used $N = 20$ and $b = 3$, therefore the individuals 8, 9, 10 are exchanged with the individuals 11, 12, 13, when the individuals are numbered from 1 to N.

An offspring is produced by modifying separately with a certain probability p each $action \in \{nextstate, setcolor, move, turn\}$:

$action \leftarrow (action + 1)\ mod\ N_{action}$.

We restricted the number of states to $N_{states} = 6$, and used $N_{setcolor} = 4$, $N_{move} = 2$, and $N_{turn} = 4$. We tested different probabilities, and we achieved good results with $p = 35\%$.

(a)


(b)


input	/	x=	0			\	/	x=	1			\	/	x=	2			\
state	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
index j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
nextstate	0	5	1	3	1	0	5	1	3	1	1	1	4	2	3	4	1	5
setcolor	2	2	1	3	2	3	3	2	2	0	2	2	2	2	0	3	2	2
move	1	0	1	0	1	1	0	1	1	0	1	1	0	1	0	1	1	1
turn	0	2	2	2	3	0	3	1	2	0	0	1	1	3	1	2	3	1

input	/	x=	3			\	/	x=	4			\	/	x=	5			\
state	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
index j	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
nextstate	2	3	1	0	2	5	1	4	0	2	0	2	0	0	4	1	2	0
setcolor	0	2	2	2	0	2	2	1	2	3	2	2	2	0	2	1	3	0
move	1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	1	0
turn	2	3	1	1	3	3	3	2	3	1	1	0	3	3	0	1	1	0

input	/	x=	6			\	/	x=	7			\	/	x=	8			\
state	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
index j	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
nextstate	3	5	5	4	5	0	4	5	4	2	3	2	2	1	2	1	3	1
setcolor	1	3	2	2	2	2	1	2	3	1	3	2	2	1	3	2	2	1
move	1	0	1	0	0	1	1	0	0	1	1	1	1	1	0	0	1	0
turn	0	0	1	0	2	1	0	3	3	2	0	3	3	0	3	2	0	3

Fig. 2. (a) Finite state machine (FSM). The state table defines the next control state, the setting of the color, the agent's new direction, and whether to move or not. (b) Example for a state table. It represents the best found, near optimal TopFSM(16) for a 8 x 8 field with 16 agents.

The fitness of our multi-agent system is defined as the number t of time steps which is necessary to emerge successfully a target pattern with a given degree h_1 of order, averaged over all given randomized initial configurations (color distribution, position and directions of the agents). As the behavior of the whole system depends on the behavior of the agents, we search for the agents' FSM that can solve the problem successfully with a minimum number of steps for a large number of initial configurations. Successfully means that a target pattern with $h \geq h_1$ was found. The fitness function F is evaluated by simulating the agent system with a tentative FSM on a given initial configuration.

$$F(FSM, config) = \begin{cases} TimeSteps & \text{if } \textit{successful} \text{ within } TimeLimit \\ HighConstant & \text{otherwise} \end{cases} \tag{1}$$

Then the mean fitness $\overline{F}(FSM)$ is computed by averaging over all given initial configurations. The mean fitness $\overline{F}$ is then used to rank and sort the FSMs. The parameters used were $TimeLimit = 5,000$ and $HighConstant = 100,000$. The genetic procedure starts with $N = 20$ random FSMs. Usually there is no FSM in the initial population that is successful. After some generations, some successful FSMs are found. Then, after further generations, FSMs are expected to be evolved that are completely successful on all or most of the given initial configurations. It turned out, that it is very difficult and time consuming to find good solutions. Therefore the genetic procedure was divided into several phases with increasing difficulty. (1) The system with $k = 16$ agents was optimized (1a) on 10 initial configuration (may also be called *field* for short) with an increasing degree of order ($h_1 = 30, 40, 48$), then (1b) on 20 fields with $h_1 = 48$, and then (1c) on 100 fields with $h_1 = 48$. Then (2 .. 7) the other systems with $k = 1, 2, 4, 8, 32, 48$ were optimized in the same way. Note that the highest degree of order is $h_{max} = 60$ for the 8 x 8 field. Thus the aimed relative degree of order was $h_1/h_{max} = 48/60 = 80\%$. Experiments showed that a higher h can be

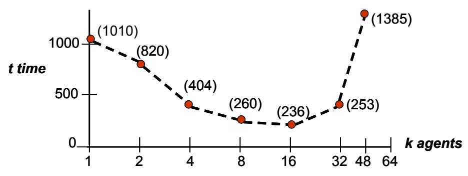


Fig. 3. Time to form the desired patterns with an 80% degree of order by the TopFSMs specifically evolved for $k = 1$ .. 48 agents. Field size is 8 x 8.

reached sometimes, but not always within the limited capabilities of the agents, the system, and the time limit. – 4500 generations (simulations with 10 mutated FSMs on up to 100 fields) were computed for each k. The overall computation time on a processor Intel Xeon QuadCore 2 GHz was around two weeks. The implementation language was Object Pascal under the platform Lazarus.

The Evolved top FSMs. The best found FSM for 16 agents (TopFSM(16)) is shown in Fig. 2b. All found TopFSM(k) for $k = 1, 2, 4, 8, 16, 32, 48$ are able to form a target pattern with an 80% degree of order for all 100 random initial configurations. The time to form the patterns reaches a minimum for 16 agents (Fig. 3), and a maximum for 48 agents. No solution within the constraints was found for 64 agents (fully packed, agents cannot move – only turn). It can be observed that a certain density of the agents (here 25% = 48/64) is optimal in order to reach the minimal time. It can be concluded that an agent can work only optimal if it is responsible for a certain "personal" local area (here 4 cells). One explanation could be that an agent uses the colors in its neighborhood in a stigmergic way (like pheromones, indirect communication mainly with itself – self-feedback memory) to be at last successful. If the field is overcrowded or sparsely populated, the performance is lowered. It was also interesting to observe, that the task can even be solved by one agent only. From the cost investment point of view ($cost = time \times agents$), one agent alone is most economical.

Simulations. (Fig. 4) The evolved TopFSM(k) were simulated and visualized. The snapshots show how the path patterns are being built. It can be seen that at the end two colors are dominating (red and yellow). The other two colors rapidly decrease and may disappear at the end. In the optimization procedure and simulation it was not forbidden to reduce the number of colors (to require an almost even color distribution of the 4 colors at the end is a too hard condition as experiments have shown). It can be seen, that the agents prefer to move on a path with the same color. Note that the degree of order was set to $\geq 80\%$. When this limit is surpassed the simulation is successful and terminates.

Versatility Test. (Fig. 5) This test was undertaken in order to prove that agents, which are optimized for special field, can be successful on another given (here larger) field with another number of agents (here larger). The test was really successful and showed that agents (as usually) can cope with different environments. The original field was of size 8 x 8, with 16 agents. The test field was of size 16 x 16, with 64 agents (the same density as in the original field).

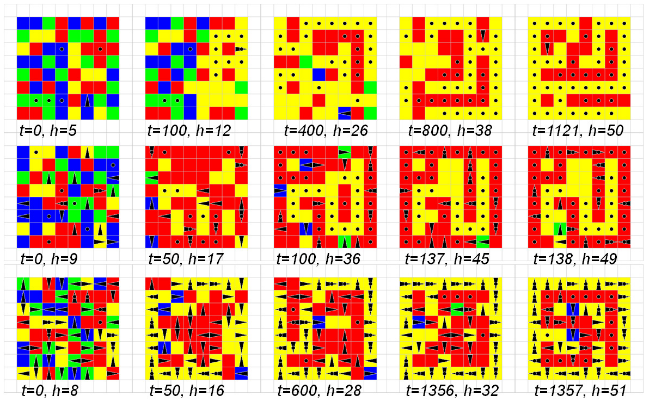


Fig. 4. Snapshots showing how path patterns are formed by k agents; $k = 1$ (top row), 16 (middle row), 48 (bottom row). h = degree of order. Dots represent template hits.

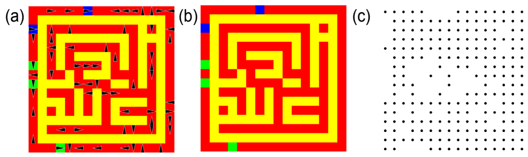


Fig. 5. The TopFSM(16) evolved on a 8 x 8 field can also solve the problem on a 16 x 16 field. (a) Agents and colors, (b) colors only, (c) template hits.

The degree of order was also set to 80%. The mean time to successfully form a path pattern for the larger field was $t_2 = 1259$, averaged over 100 random initial configurations. For comparison, the mean time on the original field was $t_1 = 236$. The ratio of the field areas is 4, and the ratio $t_2/t_1 = 5.33$, which is not extremely more.

5 Conclusion

The objective was to find FSM controlled agents that can form specific path patterns. The class of path patterns was defined by a set of templates, small 3 x 3 local patterns. The number of templates that can be found in a given pattern defines the degree of order. For a 8 x 8 field near optimal FSMs were evolved for a different number of agents. The agents are able to form successfully with an 80% degree of order the aimed path patterns. The task can be solved fastest with 16 agents (density 25%), and around five times slower with one agent only. The FSM(16) evolved for a 8 x 8 field can also handle a 16 x 16 field with the same density of agents. This means that an evolved specialist can also be successful on another field size. The general result is that specific pattern generation by CA agents can be constructed in a methodic way. It would be of interest to enhance and apply this method in order to generate more complex patterns.

References

1. Shi, D., He, P., Lian, J., Chaud, X., et al.: Magnetic alignment of carbon nanofibers in polymer composites and anisotropy of mechanical properties. Journal of Applied Physics 97, 064312 (2005)
2. Itoh, M., Takahira, M., Yatagai, T.: Spatial Arrangement of Small Particles by Imaging Laser Trapping System. Optical Review 5(I), 55–58 (1998)
3. Jiang, Y., Narushima, T., Okamoto, H.: Nonlinear optical effects in trapping nanoparticles with femtosecond pulses. Nature Physics 6, 1005–1009 (2010)
4. 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)
5. Hoffmann, R., Désérable, D.: CA Agents for All-to-All Communication Are Faster in the Triangulate Grid. In: Malyshkin, V. (ed.) PaCT 2013. LNCS, vol. 7979, pp. 316–329. Springer, Heidelberg (2013)
6. Komann, M., Ediger, P., Fey, D., Hoffmann, R.: On the Effectiveness of Evolution Compared to Time-Consuming Full Search of Optimal 6-State Automata. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 280–291. Springer, Heidelberg (2009)
7. Ediger, P., Hoffmann, R.: CA Models for Target Searching Agents. Automata 2009, J. Electronic Notes in Theor. Comp. Science (ENTCS) 252, 41–54 (2009)
8. Komann, M., Mainka, A., Fey, D.: Comparison of evolving uniform, non-uniform cellular automaton, and genetic programming for centroid detection with hardware agents. In: Malyshkin, V. (ed.) PaCT 2007. LNCS, vol. 4671, pp. 432–441. Springer, Heidelberg (2007)
9. Mesot, B., Sanchez, E., Peña, C.-A., Perez-Uribe, A.: SOS++: Finding Smart Behaviors Using Learning and Evolution. In: Artificial Life VIII, pp. 264–273. MIT Press, Cambridge (2002)
10. Blum, M., Sakoda, W.: On the capability of finite automata in 2 and 3 dimensional space. In: 18th IEEE Symp. on Foundations of Computer Science, pp. 147–161 (1977)
11. Hoffmann, R.: Rotor-routing algorithms described by CA–w. Acta Phys. Polonica B Proc. Suppl. 5(1), 53–68 (2012)
12. Hoffmann, R.: The GCA-w massively parallel model. In: Malyshkin, V. (ed.) PaCT 2009. LNCS, vol. 5698, pp. 194–206. Springer, Heidelberg (2009)
13. Hoffmann, R.: GCA–w algorithms for traffic simulation. Acta Phys. Polonica B Proc. Suppl. 4(2), 183–200 (2011)
14. Deutsch, A., Dormann, S.: Cellular Automaton Modeling of Biological Pattern Formation. Birkäuser (2005)
15. Bonabeau, E.: From Classical Models of Morphogenesis to Agent-Based Models of Pattern Formation. Santa Fe Institute Working Paper: 1997-07-063
16. Hamann, H.: Pattern Formation as a Transient Phenomenon in the Nonlinear Dynamics of a Multi-Agent System. In: Proc. of MATHMOD 2009 (2009)
17. Nagpal, R.: Programmable Pattern-Formation and Scale-Independence. MIT Artificial Intelligence Lab (2002)
18. Spicher, A., Fatèz, N., Simonin, O.: From Reactive Multi-Agents Models to Cellular Automata - Illustration on a Diffusion-Limited Aggregation Model. In: ICAART 2009, pp. 422–429 (2009)
19. Bandini, S., Vanneschi, L., Wuensche, A., Shehata, A.B.: A Neuro-Genetic Framework for Pattern Recognition in Complex Systems. Fundam. Inform. 87(2), 207–226 (2008)
20. Junges, R., Klügl, F.: Programming Agent Behavior by Learning in Simulation Models. Applied Artificial Intelligence 26(4), 349–375 (2012)