PARTICLE ALIGNMENT BY MOVING AGENTS*

Rolf Hoffmann

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

(Received March 25, 2014)

Initially, a 2D field is given that contains particles with randomly distributed spins (colors, orientations). Four different spins are assumed. The vector sum of all spins can be interpreted as a magnetic field (magnetization). The task is to align the particles by moving agents into the color which is in the majority at the beginning. First, the capabilities of the agents (actions, inputs, number of control states) were defined, because they decide on how effective the task can be solved at all. The agents' behavior is determined by an embedded finite state machine (FSM, algorithm) with 4 states only. For a given 16×16 field with 8 agents, an FSM was evolved by a genetic procedure based on mutation. The best evolved algorithm was successful for 91 out of 100 given initial fields. It turned out that the reached color alignments were not stable. After the whole field has been colored in one color, the agents proceeded to change the field into the next color. Two experiments showed that the amplitude of the magnetization decreased and the frequency increased when the number of agents was increased. The whole system including the agents was modeled by cellular automata. For the simulation of the system, the CA-w model (cellular automata with write access) was used in order to simplify the program and speed up the simulation.

DOI:10.5506/APhysPolBSupp.7.291
PACS numbers: 07.05.Tp, 32.10.Dk, 85.70.–w, 41.50.+h

1. Introduction

Given is a field of cells with particles that may have four possible orientations (spins, colors). The task is to align all the particles to the orientation which is in the majority at the beginning. The sum of all spin vectors of the whole field can be interpreted as a physical field, here called *magnetization*.

* Presented at the Summer Solstice 2013 International Conference on Discrete Models of Complex Systems, Warszawa, Poland, June 27–29, 2013.

The idea is to turn the particles by agents that are moving around. This task can be interpreted in physics as the process of spin/dipole/nanotube alignment by the use of nano robots, an external magnetic field, or by beaming certain focused waves or energy onto the cells of the array [1–3].

What is the benefit to use agents to solve this task? Generally speaking, agents can behave very flexible because they have a certain intelligence. Important properties that can be achieved by agents are:

— **Scalable**. The problem can be solved with a variable number of agents, and faster or better with more agents.

— **Tunable.** Depending on the agent's intelligence, the problem can be solved faster or more effective (better quality of solutions).

— **Adaptive.** Similar problems can be solved by the same agents, *e.g.* by changing the shape or size of the environment.

— **Fault-tolerant.** When obstacles are introduced or not all agents work correctly, the problem can also be solved in an adequate 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.

The here investigated alignment task is related to the density classification task (DCT) [4, 5]. The DCT is to decide whether the initial configuration contains more ones or zeros. It was shown that the 1D density classification is a difficult problem and that it cannot be solved by uniform cellular automata (CA) [6]. Problem solutions were proposed using non-uniform or programmable CA [7].

Our problem is more complicated, because the number of colors is four. Generally speaking, many variations of the problem are possible, *e.g.* by changing the number of colors, the dimension of the field, the shape and size of the field, or the boundary condition (cyclic, border). In addition, our problem differs from the classical CA problem because it shall be solved by moving agents. Agents can perceive the local environment and can perform certain actions (change the color, move, turn). The behavior of our agents shall be controlled by a finite state machine (FSM, control algorithm).

In former investigations, we have tried to find the best algorithms for the *Creature's Exploration Problem* [8], in which the agents have the task to visit all empty cells in shortest time, and the *All-to-All Communication Task* [10], in which each agent has to distribute its information to all the others. The FSMs for these tasks have been evolved using, *i.e.*, genetic algorithms, genetic programming [11], sophisticated enumeration methods [12],

and time-shuffling techniques [13]. In contrast to our first investigation of the problem [14], we are trying now to solve the problem with 4 control states only (instead of 10 before).

Other related work are a multi-agent system modeled in CA for image processing [15], and modeling the agent's behavior by an FSM with a restricted number of states and the evaluation by enumeration [16].

2. The problem: alignment of particles

Given is a square field of $N \times N$ cells with a border. Each cell, except the border cells, contains a particle with a certain *color* $\in \{0, 1, 2, 3\}$. The colors correspond to the orientations of the particles (toN, toE, toS, toW). The number of particles with a certain color i is E_i, all colors summing up to $N^2 = \sum E_i$. At the beginning, at time $t = 0$, the colors are randomly distributed. The goal is to align all particles to the color which has the highest frequency at the beginning (the *major color*). The alignment procedure shall be performed by $k \leq N^2$ agents that move around in the field and are able to change the particles' colors. The objective is to find the behavior of the agents that can solve the task. The capabilities of the agents shall be limited, *e.g.* the number of control states, the action set, and the details of the environment that can be perceived.

3. Modeling of the multi-agent-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 f. The rule changes the state of each cell by taking into account the own cell's state and the states of its neighbors

$$CellState(t+1) = f(CellState(t), NeighborsStates(t)) .$$

Nevertheless, the cell's rule has to react on different situations, *e.g.* whether there is an agent situated on a cell or not. Therefore, the cell's state is modeled as record comprising a type tag

$$(Type, Color, Agent) .$$

where

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

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

When designing a system with agents (multi-agent system MAS), then the capabilities of the agents have to be defined at the beginning 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 (input mapping Fig. 3, described later)

— the **own color** C of the cell the agent is situated on,

— the **color in front** C_{F} (in viewing/moving 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**: $m \in \{0,1\} = \{wait, go\} = \{M_0, M_1\}$,

— **turn**: $turn \in \{0,1,2,3\} = \{T_0, T_1, T_2, T_3\}$. The new direction is $D(t+1) = (D(t) + turn)\ mod\ 4$.

— **rotate color**: $rotate \in \{0,1,2\} = \{R_0, R_1, R_2\}$. The new color is $C(t+1) = (C(t) + rotate)\ mod\ 4$ if $rotate < 2$, and is $C(t+1) = (C(t) + rotate + 1)\ mod\ 4$ if $rotate = 2$. This means that the color can be rotated to the right (R_1), to the left (R_2), or can remain unchanged (R_0).

The move, turn and rotate actions can be performed simultaneously (24 combinations). There is a constraint for the move action: when the agent's action is *go* and the situation is *blocked*, then the agent cannot move and has to wait. In the case of a moving conflict, the agent with the lowest identifier (ID $= 0 \ldots k-1$) 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 (Fig. 1). In the 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 (the first compute the moving condition, the second

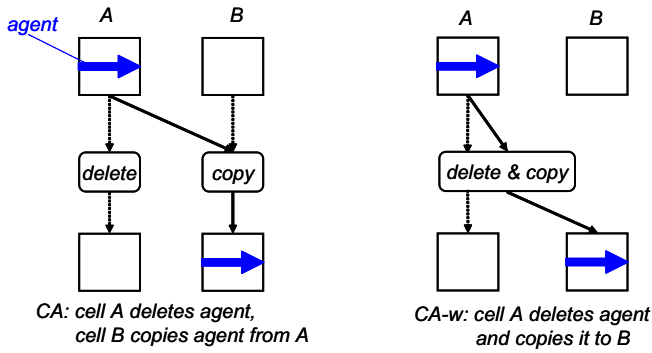

Fig. 1. In the CA model, cell A deletes the agent and cell B copies it. In the CA-w model, cell A is deleting and copying the agent.

use it in cell A and B), or use the *cellular automata with write-access model* (CA-w) [17, 18]. 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. Therefore, the CA-w model makes it easier to describe and simulate systems with moving particles. 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 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, rotate) 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

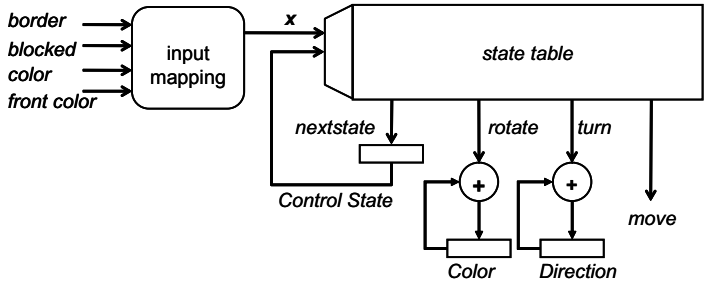

Fig. 2. Finite state machine (FSM). The state table define the next control state, the rotation of the color, the agent's new direction, and whether to move or not.

the actual line of the state table. The used input mapping is shown in Fig. 3. If the situation is *free*, the index x is the difference of the color in front and the own color: $x = C_F - C \ mod\ 4$. 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 a prior agent in the case of a conflict, then the own color is directly mapped to the index with an offset $x = C + 5$.

			x address input of state table
free	front color C_F – own color C	$C_F - C = 0$	0
		$C_F - C = 1$	1
		$C_F - C = 2$	2
		$C_F - C = 3$	3
blocked	border in front		4
	agent in front or no priority in conflict situation	C = 0	5
		C = 1	6
		C = 2	7
		C = 3	8

Fig. 3. Input mapping. The relevant input situations are mapped to a sequence of addresses to be used as inputs for the state table.

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 color.

Note that the achievable system's performance depends on the number of agents and their capabilities, *e.g.* the observable inputs, the input mapping, the defined actions, the agent's memory capacity, and the used control algorithm.

4. Evolving the agent's behavior by a genetic procedure

The ultimate objective is to find the optimal behavior on average for all possible initial configurations. As we cannot optimize for all initial configurations within a limited amount of computation time, we restrict our objective to find a near optimal behavior of the agents' behavior for a field size of 16×16 with 8 agents.

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 $K = (|s||y|)^{(|s||x|)}$, where $|s|$ is the number of control states, $|x|$ is the number of different input values 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. 4). A column j is identified by a certain combination of (x, s). Each column j defines $(s', y) = (nextstate, rotate, move, turn)$.

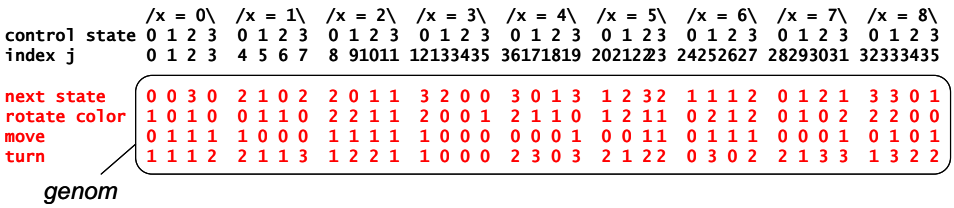

Fig. 4. State table of an FSM. It defines the behavior of an agent. The shown table represents also the best found, near optimal topFSM.

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.

We experimented also with the classical crossover/mutation method. Then we found that mutation only gave us similar good results. Therefore, we are using here only mutation. It is subject to further research which heuristic is best to evolve state machines. In previous work [19], also crossover and parallel populations were used, but at the moment we have no reliable comparisons between different heuristics to evolve state machines.

An offspring is produced by modifying separately with a certain probability the *nextstate* action, the *rotate color* action, the *move* action, and the *turn* action for each column j of the table

$$\begin{array}{lll} nextstate & \leftarrow nextstate + 1 \ mod \ N_{states} & \text{with probability } p_1 \\ rotate & \leftarrow rotate + 1 \ mod \ N_{rotate} & \text{with probability } p_2 \\ move & \leftarrow move + 1 \ mod \ N_{move} & \text{with probability } p_3 \\ turn & \leftarrow turn + 1 \ mod \ N_{turn} & \text{with probability } p_4 \,. \end{array}$$

We restricted the number of states to $N_{states} = 4$, and used $N_{rotate} = 3$, $N_{move} = 2$, and $N_{turn} = 4$. We tested different probabilities, and we achieved good results with $p_1 = p_2 = p_3 = p_4 = 35\%$.

The fitness of our multi-agent system is defined as the number of steps which are necessary to align the particles, averaged over all given 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.

The fitness function F is evaluated by simulating the agent system with a certain FSM on a given initial configuration

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

Then the mean fitness $\overline{F}(FSM)$ is computed by averaging over the given initial configurations in the set. The mean fitness $\overline{F}$ is then used to rank and sort the FSMs. The parameters used were $TimeLimit = 15,000$ 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. The *major color* is the color with the highest frequency at the beginning. The *minor colors* are the other three colors at the beginning. The frequency of the major color was set to $P = N^2/4 + 3u$, and the frequency of each minor color was set to $Q = N^2/4 - u$. The parameter u is a measure for the deviation from the case where each color has the same frequency $N^2/4$.

In each phase, $4w$ initial fields were used with the following initial color distribution: w fields with $(E_0, E_1, E_2, E_3) = (P, Q, Q, Q)$, w fields with (Q, P, Q, Q), w fields with $(Q,\ Q,\ P,\ Q)$, w fields with $(Q,\ Q,\ Q,\ P)$.

In the first phase, 4 initial configurations with increasing difficulty (subphases with decreasing parameter u) were used, $(w, u) = (1, 40)$, $(1, 20)$, $(1, 10)$, $(1, 5)$, $(1, 3)$. In the second phase, 20 initial configurations with decreasing u were used, $(w, u) = (5, 10)$, $(5, 5)$, $(5, 3)$. In the third phase, 100 initial configurations with decreasing u were used, $(w, u) = (25, 5)$, $(25, 3)$. At last, the frequency of the main color was 73, and the frequency of the minor colors 61. At each optimization phase, the already evolved FSMs were used as input for the next phase.

The overall computation time on a processor Intel Xeon QuadCore 2 GHz was around one week. The best found FSM (topFSM) is shown in Fig. 4. The topFSM is able to align successfully 91 fields out of 100 within the given time limit of 15,000 (Fig. 5). Compared to the previously found FSM with 10 states [14], the now found FSM is equally successful but needs only 4 states. It should be noticed that the alignment is not necessarily stable (stationary) but temporary. This means that after or even before the field is correctly aligned, it may be aligned to another color (see simulations in the next section).

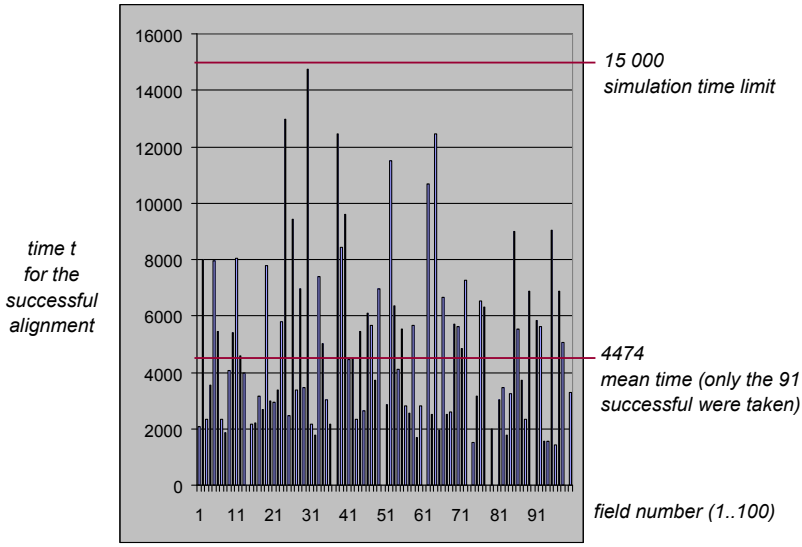

Fig. 5. Alignment time. 91 fields out of 100 were successful within the time limit of 15,000.

Programming Details. The implementation language was Object Pascal (Free Pascal) under the development platform Lazarus. The main programs are the optimizer (genetic procedure) P1 and the simulator P2. P1 and P2 each consists of around 2000 lines of code, where around 1200 lines are identical. Small additional programs are used to generate an initial set of

FSMs, and the training and test field configurations. Most of the parameters are easily changeable because they are stored in parameter files that can be modified by a text editor. Output of the simulation runs are ASCII text files. These files can be translated and compacted to a simple byte format, which can be read and visualized by a separate tool written in JAVA.

The FSMs are programmed as arrays with index addressing. Thereby, a fast access to the actions is guaranteed.

The optimizer is organized by an inner FOR-loop (T time steps) for the simulation, enclosed by three nested FOR-loops. The number MAXgeneration of optimizations varied for the different runs, between 10 and 500.

```
empty -> A'
FOR generation 1 .. MAXgeneration do
  FOR each fsm(j) in the set AB of FSMs
    FOR each given field(i)
       FOR T=0 to Time_Limit
           simulate time step for (field(i), fsm(j))
           T+1 -> T
           break loop if (field aligned)
       ENDFOR time-step
       store simulation time T(j,i)
    ENDFOR field
  ENDFOR fsm

  compute Fitness for each fsm, average T over all fields
  sort_according_to_Fitness(AB,A')
  delete_duplicates -> AB
  exchange some FSMs from second half of B with first half of A
  mutate(A)-> A'
ENDFOR generation
```

5. Simulations

The effect of aligning the particles by the agents using the evolved topFSM is demonstrated by different simulations. The first simulation on the initial field (Fig. 6) is displayed in Fig. 7. At $t = 0$, the major color is 0 (73 times). Then at $t = 953$, color 1 is most frequent (146), then at $t = 1173$ color 2 is most frequent (146), and at last at $t = 2070$, the whole field is aligned to the color 0.

In order to get more insight about the time evolution of the alignment, a phase diagram of the total magnetization and the time evolution of the magnetic field were used. Abstracting from the real physics, it was assumed that the colors represent magnetic spins with four possible directions. The

```
. . . . . . . v5. . . . . . . . .      1 1 3 1 1 1 2 2 0 0 3 0 2 2 2 0
. . . . . . . . . . . . . . . .        3 1 1 1 2 2 2 1 2 0 0 0 2 0 3 1
. . . . . . . . . . . . . . . .        0 1 1 0 3 2 1 0 3 2 1 0 3 3 2 2
. . . . . . . . . . . . <7. >3.        2 3 1 2 0 3 3 3 1 3 0 3 3 2 1 0
. . . . . . . . . . . . . . . .        2 2 3 1 2 0 0 3 3 1 0 3 0 2 1 3
. . . . . . . . . . . . . . . .        0 0 0 2 1 2 3 3 3 1 0 1 2 0 1 1
. . . . . . . v4. . . . . . . . .      2 2 0 0 2 2 3 1 2 0 1 2 3 1 3 2
. . . . . . . . . . >2. . . . . .      2 3 1 2 0 0 0 0 0 2 0 1 1 2 3
. . . . . . . . . . . . . . . .        2 0 1 0 1 3 2 0 1 0 3 3 3 1 3 3
. . . . . . . . . . . . . . . .        2 2 2 3 0 3 2 2 0 3 0 0 3 3 2 1
. . . . . . . . . . . . . . . . v0     0 0 0 0 3 0 0 2 3 0 2 0 1 3 2 3
. . . . . . . . . . . . . . . .        0 0 1 0 1 1 1 1 3 1 1 1 2 0 0 1
. . . . . . . . . . . . . . . .        1 0 3 0 0 3 1 2 2 1 1 2 0 3 1 3
. . . . . . . . . . <1. . . . . .      3 0 1 2 2 2 3 2 3 1 2 2 3 0 3 3
. . v6. . . . . . . . . . . . . .      3 2 2 1 0 2 1 3 0 0 2 0 1 1 0 0
. . . . . . . . . . . . . . . .        0 1 3 0 1 1 0 2 0 0 1 3 3 3 3 0

agents (ID = 0,1,2,3,4,5,6,7)          t=0, #colors=(73,61,61,61),
                                       73 cells have color 0
```

Fig. 6. Initial configuration at time $t = 0$. Field 1 with agents (left), colors (right).

```
2 2 2 2 1 1 0 0 0 0 0 0 0 0 0 1    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 1 1 1 1 1 1 1 0 0 0 0 0 0 1    2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1    3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1    2 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 1 1 1 1 1 1 1 0 0 1 1 0 1    1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 1 1 2 2 1 1 1 0 3 3 1 1 1    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 1 2 2 1 0 0 0 0 0 2 1 0 1    2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 2 1 2 0 0 0 0 0 0 1 1 1 1 1    2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 1 1 1 1 2 2 0 1 0 0 2 1 1 1 1    2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 2 1 1 1 1 1 1 0 0 1 1 1 1 1    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 1 1 1 1 1 0 0 0 0 1 1 1 1    2 2 2 2 1 1 1 2 1 1 1 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 1 1 1 1 1 1 1 0 0 1 1 1 1 1    2 3 2 1 1 1 1 1 1 1 1 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 0 1 0 1 1 1 1 1 0 1 1 1 1    2 2 3 3 1 1 1 1 1 1 1 2 3 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 0 1 1 1 1 1 1 1 1 0 1 1 1 1    2 2 2 3 2 1 2 2 1 1 1 3 3 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

t=953, #colors=(54,146,54,2)       t=1173, #colors=(0,28,211,17)      t=2070, #colors=(256,0,0,0)
```

Fig. 7. Simulation of the alignment to the major color 0 by eight agents with topFSM on field 1.

whole magnetic field (magnetization $\boldsymbol{M}$) shall be given by the sum of all the spin vectors in the field. The x/y-component of $\boldsymbol{M}$ is called M_x resp. M_y, where

$$\boldsymbol{M} = M_x \boldsymbol{u}_x + M_y \boldsymbol{u}_y \,,$$

where the components can be determined by the number E_i of the spins:

$$M_x = E_1 - E_3 \,, \qquad M_y = E_0 - E_2 \,.$$

The phase diagram represents the time evolution of the magnetization (vector of the magnetic field), or the relation $(M_x(t), M_y(t))$.

For the initial field (Fig. 6) and using 8 agents with the topFSM, the phase diagram of the magnetization and their components are shown in Fig. 8. At $t = 0$, the magnetization points slightly to the North, then increasingly to the South, to the West, to the North again, then to the West

(point A, all spins are aligned), and finally to the North (the target point B, all spins are aligned). And after that, the magnetization is not stable, it is circulating. The original goal was to reach the point B directly as a fix point, and not on an orbit. But we were not able to reach this goal with the limited capabilities of the agents. Nevertheless, the result was interesting because the target point was reached on an orbit in 91% of the test cases. If the task of the agents would have been to produce an oscillating magnetization, then the agents had been very successful.

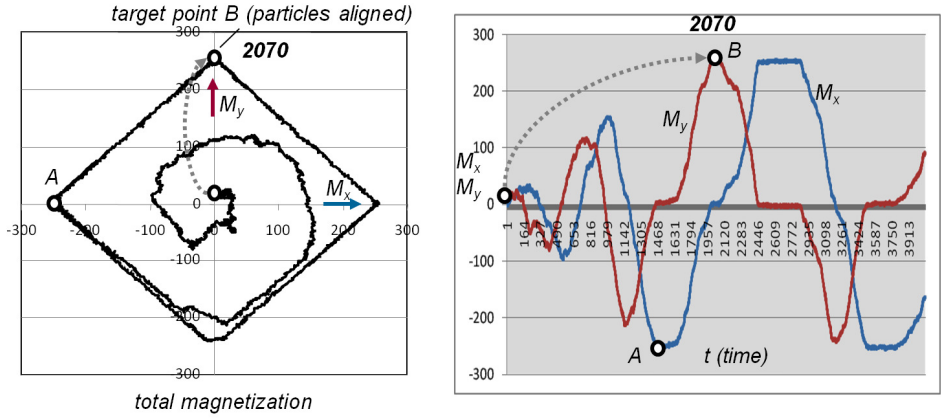

Fig. 8. Phase diagram of the total magnetization (left) and time evolution of the magnetization components M_x and M_y (right). First all spins are aligned to East (point A), then at time 2070 all spins are aligned to North (point B). At the beginning, the major color was 0 (toNorth). Field size 16×16, 8 agents.

There were two other experiments performed in which the number of agents were increased in order to test for scalability. From other tasks with agents we know that often the task is solved faster and also successfully with more agents. When using 16 agents on the same initial field and the same topFSM (Fig. 9), the magnetization is also oscillating but with a higher frequency. The highest reached amplitude that was before 256, was now $\max(|M_x|, |M_y|) = 254$.

When using 256 agents, meaning that the system is fully packed with agents (Fig. 10), the frequency of the oscillations was even higher and the amplitude lower (175). The shape of the M_x and M_y signals appear as polygons with small random variations.

These two experiments have shown that agents may be used to generate oscillating signals with a certain shape and with some random variations. Thus other FSMs for agents could be evolved that are optimized to produce special formed signals, like sinus waves or random noise.

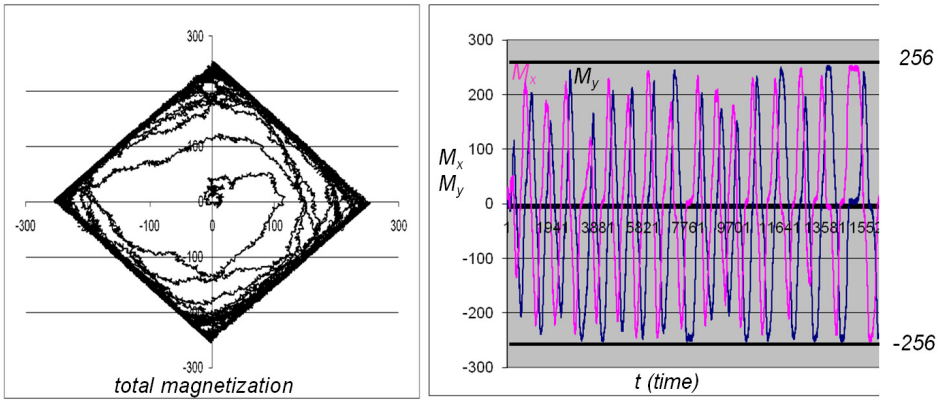

Fig. 9. Experiment with 16 agents. Phase diagram of the magnetization (left) and time evolution of their components (right). The generated signals are oscillating, their shapes and amplitudes are noisy.

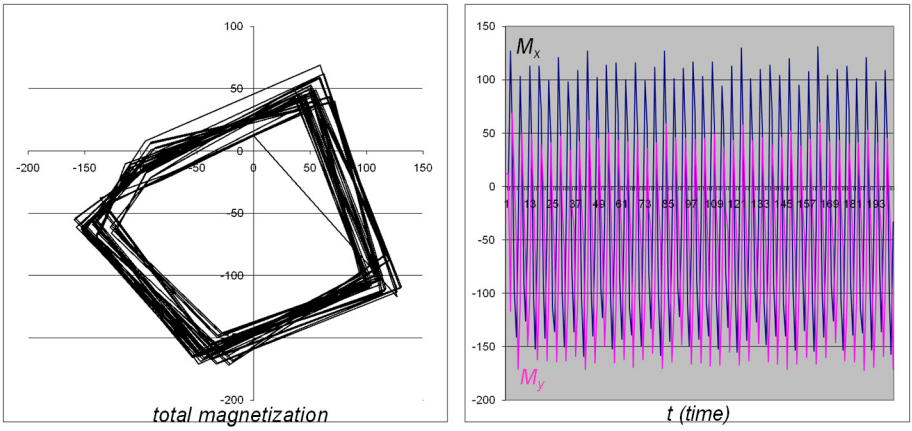

Fig. 10. Experiment with 256 agents (fully packed). Phase diagram of the magnetization (left) and time evolution of their components (right). Now the signals' amplitudes are lower but the frequency is highest.

6. Conclusion and future work

First the capabilities of the agents were defined, because they decide on how effective the task can be solved. The defined agents can perform 24 actions, combinations of moving, turning and coloring. They can react on 9 input situations, combinations of the own color, the color in front, the own direction, and blocking cases. The agents' behavior is determined by an embedded finite state machine (FSM) with 4 states only. For a given 16×16 field with 8 agents, an FSM was evolved by a genetic procedure based

on mutation, and the procedure was executed stepwise with an increasing difficulty and an increasing number of test fields. The best evolved algorithm can perform the task for 91 out of 100 given fields. The reached mean time was 4,474 for the 91 successful fields. Although originally it was intended to reach the target point directly as a fix point (stable maximum of the total magnetization in the direction of the initial major color), it turned out that the magnetization rotates, thereby passing the target point cyclically. This means that the task could not be completely solved with the given capabilities. It is an open question if there exists a 4-state algorithm, which reaches the target point directly for a high percentage of initial fields and does not generate a cyclic magnetization. The scalability (changing the number of agents) was also tested using 16 and 256 agents. It turned out, that by increasing the number of agents, the amplitude of the magnetization decreases and the frequency of the oscillations increases.

Future work is aimed to find more powerful and more intelligent agents that can solve the problem better or perfectly. It is also very interesting to find agents that can produce oscillations of the magnetization with a certain frequency and shape, or pseudo-random noise.

REFERENCES

[1] D. Shi *et al.*, *J. Appl. Phys.* **97**, 064312 (2005).

[2] M. Itoh, M. Takahira, T. Yatagai, *Opt. Rev.* **5**, 55 (1998).

[3] Y. Jiang, T. Narushima, H. Okamoto, *Nature Phys.* **6**, 1005 (2010).

[4] S. Verel, P. Collard, M. Tomassini, L. Vanneschi, *Lect. Notes Comput. Sci.* **4173**, 258 (2006).

[5] P. Oliveira, *Lect. Notes Comput. Sci.* **8155**, 1 (2013).

[6] M. Land, R.K. Belew, *Phys. Rev. Lett.* **74**, 5148 (1995).

[7] S. Sahoo, P. Choudhury, A. Pal, `arXiv:0902.2671 [nlin.CG]`.

[8] M. Halbach, R. Hoffmann, L. Both, *Lect. Notes Comput. Sci.* **4173**, 571 (2006).

[9] P. Ediger, R. Hoffmann, *Lect. Notes Comput. Sci.* **5698**, 182 (2009).

[10] R. Hoffmann, D. Désérable, *Lect. Notes Comput. Sci.* **7979**, 316 (2013).

[11] M. Komann, P. Ediger, D. Fey, R. Hoffmann, *Lect. Notes Comput. Sci.* **5481**, 280 (2009).

[12] M. Halbach, Algorithmen und Hardwarearchitekturen zur optimierten Aufzählung von Automaten und deren Einsatz bei der Simulation künstlicher Kreaturen, Dissertation Technische Universität Darmstadt, 2008.

[13] P. Ediger, R. Hoffmann, Evolving Hybrid Time-shuffled Behavior of Agents, IEEE International Symposium on Parallel & Distributed Processing, Workshops and Ph.D. Forum (IPDPSW), 2010, pp. 1–8.

[14] P. Ediger, R. Hoffmann, Alignment of Particles by Agents with Evolved Behaviour, Workshop Innovative Rechnertechnologien, Nanotechnologien für die IT: Juli 9–10, 2009, Helmut-Schmidt-Univ., Germany.

[15] M. Komann, A. Mainka, D. Fey, *Lect. Notes Comput. Sci.* **4671**, 432 (2007).

[16] B. Mesot, E. Sanchez, C.-A. Peña, A. Perez-Uribe, SOS++: Finding Smart Behaviors Using Learning and Evolution, Proc. of the 8th International Conference on Artificial Life, pp. 264–273, 2002.

[17] R. Hoffmann, *Acta Phys. Pol. B Proc. Suppl.* **3**, 347 (2010) .

[18] R. Hoffmann, *Acta Phys. Pol. B Proc. Suppl.* **4**, 183 (2011).

[19] P. Ediger, Modellierung und Techniken zur Optimierung von Multiagentensystemen in Zellularen Automaten, Dissertation, TU Darmstadt, Darmstadt, Germany, 2011.