

CA Models for Target Searching Agents

Patrick Ediger¹ Rolf Hoffmann²

*FB Informatik, FG Rechnerarchitektur
Technische Universität Darmstadt
Germany*

Abstract

Agents in a cellular grid have the task to move from their start positions to their individual target positions as fast as possible. Four models using agents are proposed that can be applied to the problem. These models are conform to the CA paradigm. The agents have either a moving direction (directed agent) or not (undirected agent). The agents behave either in a deterministic way according to a control automaton inside of each agent or they behave randomly. In order to find the best behaving agents, control automata (“algorithms”) were evolved using a genetic island model. Near optimal algorithms were evolved separately for $k = 1$ to $k = 256$ agents in a 32×32 environment using 20 random initial configurations for each k . Then these algorithms were ranked using another set of 100 initial configurations for each k . It turned out that the agents behave better with respect to speed and reliability in this order: (1) controlled directed agents, (2) random directed agents, (3) random undirected agents, and (4) controlled undirected agents. Although the controlled directed agents (optimized for each k) can solve all the given 100 initial configurations in the ranking set, it can not be assured that no deadlocks may occur for other initial configurations.

Keywords: CA agent modeling, multi-agent target search, evolving behavior of agents, intelligent agents

1 Introduction

In general our research is directed to model applications with agents within the CA model, and to find automatically the optimal behavior of the agents in order to solve a given task.

Let us consider a cellular grid with k agents. Each agent i has the task to move from its initial position (u_i, v_i) to its target position (r_i, s_i) stored in the agent as relative coordinates ($\Delta x = r_i - u_i$, $\Delta y = s_i - v_i$). The whole task is to reach all the targets as fast as possible. When an agent reaches a target, it is deleted. At most one agent can be located on one site. Each agent has its own target and the targets are disjoint. Thus each agent tries separately to move to its own target. When the agents are moving to their targets they will constrain each other by cells that are

¹ Email: ediger@ra.informatik.tu-darmstadt.de

² Email: hoffmann@ra.informatik.tu-darmstadt.de

selected in common (conflicting cells). Note that agents may run into deadlocks which implies that some agents can not reach their targets.

The goals of this investigation are twofold:

- (1) Find a CA modeling of the problem that is promising, meaning that in general the goal (2) can be achieved well.
- (2) Find an optimal CA rule that (a) the problem is solved for a maximum percentage of all combinations of the possible initial and target positions and (b) with a minimum number of generations (maximal speed).

In order to pursue the goal (1), four models were designed (see section 2 for details):

- (i) (Model M1) An agent can move to any of the four orthogonal directions. The actual chosen direction during the current generation is determined by a *control automaton* inside the agent. This type of agent is denoted as “*undirected agent*”.
- (ii) (Model M2) An agent has no direction but chooses randomly one direction that leads directly to the target, i. e., that assures to come closer to the target.
- (iii) (Model M3) An agent has a certain moving direction. The next direction (out of four) is determined by a *control automaton* inside the agent. This type of agent is denoted as “*directed agent*”.
- (iv) (Model M4) An agent has a certain moving direction. The next direction (out of two, shortest towards the target) is determined randomly.

In order to meet the goal (2), optimal CA rules were evolved using a genetic algorithm. A CA rule (single cell CA automaton) is composed of a fixed part and a configurable part. The configurable part is a control automaton that reacts on inputs and that determines the actions. Note that all the models are completely conform to the CA paradigm using a Manhattan neighborhood distance of two or three.

The task of the whole agent system can be seen as transporting individual messages to individual targets. Note that it is not assumed that the set of initial positions I is disjoint from the set of target positions T . A special case is given when the set of initial positions is equal to the set of target positions ($I = T$). Then the agents perform a permutation of their positions. - The proposed models will also work when a group of agents have the same target (e. g., all agents shall move to the same sink). But we will assume in this investigation that the targets are disjoint in order to simplify the problem, i. e., to reduce the number of initial configuration settings.

A CA based path planning algorithm in multi-agent systems has been proposed in [17], where many agents have to find the same target. Our investigation is related to this work, but a main difference in our task is, that each agent has its own individual target. The target search in agent systems has been researched in many variations: with moving targets [11,4], and in single-agent systems [10]. Here we restrict our investigation to stationary targets, and multiple agents having only

a local view.

In former works we investigated multi-agent systems in CA with different tasks, like the Creature’s Exploration Problem [6] or the All-to-All Communication task [1]. In these investigations we used different methods of optimization like genetic programming [8], genetic algorithms [2], sophisticated enumeration [5] and time-shuffling techniques [1].

A transactional CA model for multi-agent systems was developed in [16]. In general our work is also related to works like: Evolving optimal rules for CA [14,15], finding out centroids with marching pixels [9], simulation of pedestrian behavior [12] or traffic flow [13].

The remainder of this paper is organized as follows. In Section 2 we describe the four models M1 to M4 in more detail. Section 3 describes the optimization technique for the agents’ behavior. The quality of all four models is evaluated in section 4 and in section 5 the paper ends with a conclusion.

2 The Four Models

The first idea is to use a specific (“implicit”) CA rule to solve the problem which is not customized to this application. Such a rule has to assure that the information is transported correctly from the initial to the target positions, via an implicit network formed by the cells themselves. This network can be formed implicitly by the cell rule, or it can be organized using explicit information transfers. As it is very difficult to find implicit rules we decided for explicit rules using “agents”. In this context an agent (modeled within the CA paradigm) can carry information to its nearest NESW neighbor. An information transfer is modeled by a pair of rules: The source cell deletes the information, the neighboring cell copies the information. We did not pursue other approaches conserving the number of “information particles” (lattice gas modelling technique, or Margolus Neighborhood [18], or two-phase models in general [7]) because our models should work with explicit transfers (agent based) and without using two phases.

2.1 Model M1: Undirected agent

An undirected agent has no stored direction. It decides during the current generation in which direction out of four (NESW) it wants to go. The action taken by the agent depends on the inputs (states of neighbors within Manhattan distance of two) and the current *control state* of the control automaton inside the whole CA automaton. Apart from the state of the control automaton, the cell state includes the relative target information (Δx , Δy) and the *celltype* $\in \{AGENT, OBSTACLE, EMPTY\}$ (Fig. 1).

The whole cell acts as a MOORE machine which is standard in CA. The control unit is a MEALY machine and is a part of the whole MOORE machine.

An agent cannot move in a certain direction if (1) the neighbor cell in that direction is of the celltype OBSTACLE (border) or AGENT, (2) or a conflict is

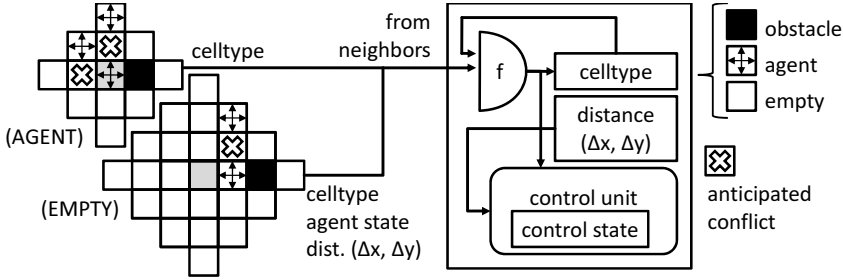


Fig. 1. The structure of a cell in the undirected model M1. The control unit is embedded in the cell structure. It comprises the control automaton with the *control state* (Fig. 2).

anticipated, which is true when another agent is located at Manhattan distance 2, because the moving direction of another agent is not known in advance.

In the case that the agent can move: The decision is taken by the control automaton residing inside the control unit (Fig. 2). The input for the control automaton are the relations ($>$, $=$, $<$) of the distances Δx and Δy . We used an input reduction table to assign discrete values to all the possible input combinations (Fig. 3). The control automaton computes internally a certain priority list (1 out of $4!$, e. g., NSEW). Then the list of free moving directions is checked. In this example: if N is free then move to N; else if S is free then move to S; else if E is free then move to E; else if W is free then move to W; else wait.

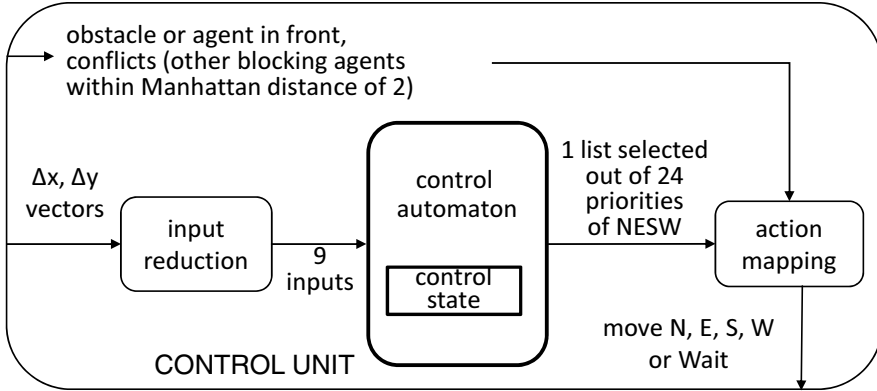


Fig. 2. Undirected agent. According to Fig. 3 ($\Delta x, \Delta y$) are converted into 9 discrete values. The control automaton computes an internal priority list. The priority list defines the actual moving direction (among the directions which are free).

The control unit realizes together with the function f the cell rule, which can be informally described as follows:

- If the celltype is EMPTY, calculate whether a conflict can be anticipated or not. If there is no conflict, use the control state and the target information ($\Delta x, \Delta y$) of the nearby agent A in the own control automaton (located in the own cell) to determine the moving decision of the agent A. For that purpose the states of all the neighbors of the agent A have to be read, thus a Manhattan distance of 3 is necessary. If the moving decision of agent A is to move to the own cell, change celltype to AGENT and update the target information by decrementing

Δx	Δy	reduced input
<0	<0	0
<0	=0	1
<0	>0	2
=0	<0	3
=0	=0	4
=0	>0	5
>0	<0	6
>0	=0	7
>0	>0	8

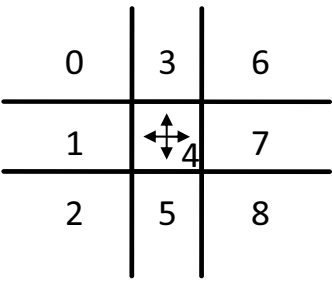


Fig. 3. The input reduction table defines how the relative position to the target is distinguished by 9 values. In this actual case, the number of inputs is not reduced.

or incrementing Δx or Δy .

- If the celltype is AGENT, detect whether there are possible movements. If there is at least one possible movement, change celltype to EMPTY.
- If the celltype is OBSTACLE, do nothing.

2.2 Model M2: Randomized undirected agent

In this model no control automaton is used. Instead the agent tries to move directly on the shortest path to the target. The shortest path is any path that reduces the distance to the target in each step in any of the two directions. Often there are two alternate directions that can be used for the shortest path. If the agent has a choice, the choice is decided randomly. If a conflict is anticipated in direction of the shortest path, the agent randomly chooses one of the other directions, or it is blocked when a conflict is anticipated in all four directions. Note that it is not necessary for success that all agents follow the shortest path, nevertheless they will follow it, if no conflicts occur.

2.3 Model M3: Directed agent

In this model the agent has a moving direction. If the agent can move it will move in that direction, and it will either move straight ahead (Sm) without turning, or will turn simultaneously to the right (Rm), to the left (Lm) or backwards (Bm). If two agents meet each other and they are pointing to each other, then they are interchanged (*swap*) and in addition their directions can be changed. In case of a conflict ($h = 2, 3$, or 4 agents meet at a crossing and point to the crossing cell) the crossing cell acts as an arbiter. There are 24 priority schemes (possible permutations) to resolve the conflict for 4 agents. The agent with the highest priority decided by the crossing wins and moves to the crossing point. E.g., if the crossing priority scheme is NSEW and one agent from S and one agent from W want to move to the crossing point, then the agent from S will win and move. The other agent stays on its cell but still performs a turning action (S (no turn), R , L or B (turn right/left/back)). The 24 possible priority schemes are equally distributed over the cell space (one after the other repeatedly) in the initial configuration.

The structure of the cell is more complex in comparison to the structure of model M1 (Fig. 4). It has the additional states *direction* and *priority*, but the neighborhood can be reduced to the Manhattan distance of 2.

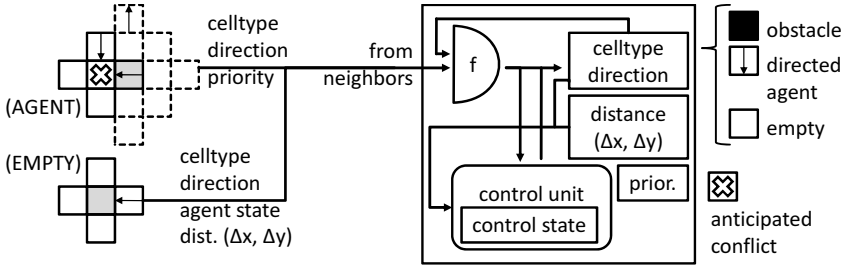


Fig. 4. The structure of a cell in the directed model M3. The control unit is embedded in the cell structure. It comprises the control automaton with the *control state*.

Like in the model M1 the decision of the agent is taken by the control automaton inside the control unit (Fig. 5). The inputs for the control automaton consist of the relations ($>$, $=$, $<$) of the distances Δx , Δy and the direction of the agent. In order to keep the control automaton simple, the 36 possible combinations are reduced to 6 discrete values (Fig. 6). The control automaton computes internally the preferred turning direction. Then the possibility of moving forward (including the swapping option) is checked.

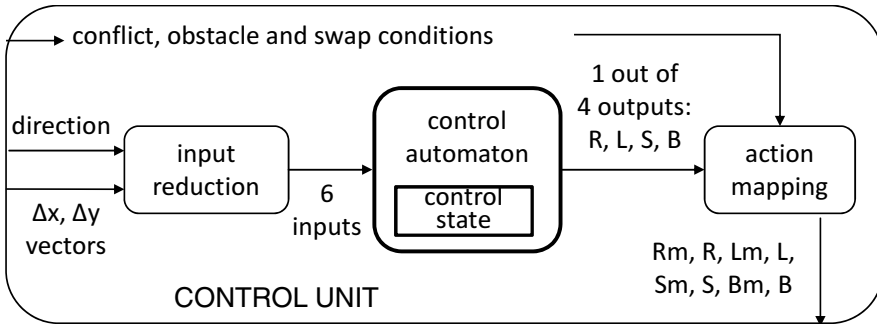


Fig. 5. Directed agent. According to Fig. 6 ($\Delta x, \Delta y$) and the current direction of the agent are converted into 6 discrete values. The control automaton computes the turning direction.

The cell rule, again realized by the function f and the control unit, can be informally described as follows:

- If the celltype is EMPTY, calculate which of the four neighboring cells with celltype AGENT pointing to the own cell has the highest priority. If such an agent exists, copy the control state, direction and the target information and use it in the own control automaton to determine (in the own cell) the turning decision of the agent. Finally change the celltype to AGENT and update the target information by decrementing or incrementing Δx or Δy .
- If the celltype is AGENT, detect whether the movement to the front cell is possible (including the swapping option and taking into account the priority list of the front cell). If it is possible, change celltype to EMPTY. Note that a neighborhood

direction	Δx	Δy	reduced input
W N	<0	<0	5
E S	<0	<0	2
N S	<0	=0	3
W	<0	=0	4
E	<0	=0	1
N E	<0	>0	2
W S	<0	>0	5
N	=0	<0	4
W E	=0	<0	3
S	=0	<0	1
*	=0	=0	0
N	=0	>0	1
W E	=0	>0	3
S	=0	>0	4
N E	>0	<0	5
W S	>0	<0	2
N S	>0	=0	3
W	>0	=0	1
E	>0	=0	4
E S	>0	>0	5
W N	>0	>0	2

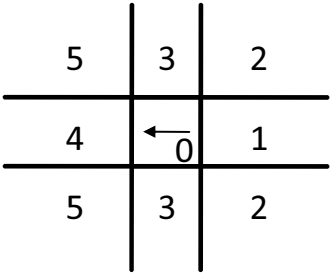


Fig. 6. The input reduction table defines how the relative position to the target is distinguished by 6 values. In this case, 36 different input combinations are reduced to 6 inputs for the control automaton.

with Manhattan distance of 2 is needed here.

- If the celltype is OBSTACLE, do nothing.

2.4 Model M4: Randomized directed agent

Each agent has a moving direction. No control automaton is used. Instead the agent tries to move directly to the target by turning itself in the right direction. Often there are two alternate next directions that can be used for the shortest path. If the agent has a choice, the choice is decided randomly. If a conflict is anticipated one agent is selected to move by the arbiter comprised in the crossing cell (as in the model M3). If two agents are in direct contact and point to each other they are swapped (as in the model M3).

3 Evolving control automata

A fixed 2D grid of 32 × 32 cells with border was chosen in order to restrict the parameter space. The task was investigated separately for the four models and separately for k = 1, 2, 4, 8, 16, 32, 64, 128, 256 agents.

For the models M1 and M3 near optimal control automata (also called "algorithms" in this context) were evolved using an island model genetic algorithm as described in [2,3]. In each run, three islands were used with a population size of 100 automata (initially randomly generated) and an immigration rate of 2%. During the evolution process, the new automata were constructed using a uniform crossover

with two parent automata like in [2,3].

First Step. For each k , 20 random initial configurations were used as a *Training Set* to evolve algorithms (MEALY FSMs, control automata). The initial positions (and the directions for models M3 and M4) of the agents and the target positions were placed randomly. For each k , 6 runs of the genetic algorithm were performed. The number of control automata states was restricted to 6. The fitness function is a dominance relation:

1. Number of configurations in which the agents reach all targets within 10,000 time steps
2. Number of not reached targets after 10,000 generations
3. Number of generations needed to reach all targets

The results of the 6 runs were merged and ranked to Top 100 lists.

Second Step. In order to find out the algorithms that behave best on any initial configuration (not only on the Training Set), another *Ranking Set* of initial configurations was used. For each k , 100 configurations were randomly computed (two different sets for the directed and the undirected model). Then each Top 100 list was reordered according to their fitness (averaged over the 100 configurations of the Ranking Set). The result was a Top 10 (k) list for each k and model. In order to evaluate the randomized models M2 and M4, the same 100 random initial configurations as for the models M1 and M3 were simulated 10,000 times (in total 1,000,000 simulations) for each k .

4 Evaluation

For a given initial configuration the minimum number of generations that can be reached is the longest distance of all Manhattan distances between the initial and target positions. Due to conflicts and congestion this limit is not always reachable especially if the number of agents is growing over a certain limit. In addition deadlocks may appear that can make the whole problem unsolvable. The deadlock problem is interesting but cannot be discussed here further.

4.1 Undirected agents (M1)

We expected that the undirected agents will behave better than the directed ones. But the opposite is true. The undirected agent (with the Top 1 algorithm) is completely successful (for all the 100 configurations in the Ranking Set) for $k = 1, 2, 4, 8$ agents (Table 1), but the algorithms for $k > 8$ are slow and/or not completely successful (Table 1, see also Fig. 7).

4.2 Randomized undirected agents (M2)

This agent behaves better than the agent M1 for $k = 4, 8, 16$ (Fig. 7). For $k = 32, 64, 128, 256$ it is only partially successful (99.99%, 99.68%, 97.64%, 71.7%). In both models M1 and M2 the number of time steps increases strongly for $k > 16$ or

Table 1

Undirected Agents. The Top 2 evolved FSMs for a given k (shaded diagonal). These algorithms were also used for a robustness check using a different number of k (non diagonal elements). The bottom row d_M denotes the maximal initial Manhattan distance of all agents to their target, averaged over all 100 configurations of one set.

	Set k=1		Set k=2		Set k=4		Set k=8		Set k=16		Set k=32		Set k=64		Set k=128		Set k=256	
	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps
Top2	100	23.03	100	27.36	95	35.19	74	37.59	31	48.65	2	55	0		0		0	
k=1	100	23.03	100	27.6	95	36.34	78	39.65	27	55.78	0		0		0		0	
Top2	100	24.55	100	29.43	98	38.48	96	48.53	76	59.71	39	73.95	0		0		0	
k=2	99	27.71	100	33.16	96	42.54	81	48.51	43	63.74	1	88	0		0		0	
Top2	100	30.91	100	36.24	100	45.71	94	59.52	66	68.61	16	85.88	0		0		0	
k=4	100	31.71	100	39.22	100	48.25	98	57.51	83	71.27	49	83.41	4	99.75	0		0	
Top2	100	23.41	100	27.73	100	34.64	100	38.39	96	44.88	60	55.85	6	73.83	0		0	
k=8	100	23.47	100	27.98	100	34.86	100	39.31	95	47.12	78	57.08	23	85.48	0		0	
Top2	100	44.79	100	55.79	100	67.89	100	84.63	99	94.75	99	111.2	86	133.7	45	239.3	0	
k=16	100	41.71	100	54.81	100	74.33	100	85.38	99	101.9	86	119.9	38	141.3	0		0	
Top2	100	68.85	100	93.03	100	123.8	100	160.7	100	189.6	100	217.4	90	256.7	39	384.8	0	
k=32	100	61.95	100	81.46	100	110.3	100	149.2	99	177.6	98	203.7	83	235.3	37	369.3	0	
Top2	100	50.63	100	64.14	100	78.32	100	113.5	100	132.9	97	173.9	76	214.3	20	420.3	0	
k=64	100	57.43	99	66.19	97	89.12	97	112.6	88	127.3	83	160.1	62	206.1	11	345.7	0	
Top2	95	41.23	93	50.72	88	65.33	75	70.03	62	95.68	47	123	36	181.3	43	387	5	2798
k=128	100	58.29	100	69.85	100	84.35	95	99.4	83	122	66	158	47	234.7	36	405.1	0	
Top2	41	41.46	19	50.84	3	80	0		0		0		0		0		0	
k=256	37	110.3	10	126.2	1	160	0		0		0		0		0		0	
max(d_M)		23.03		27.28		33.72		36.52		41.48		45.79		48.63		51.07		52.65

$k > 64$. This is due to the congestion which is more likely to appear when more agents move on the 32×32 grid.

Table 2

Directed Agents. The Top 2 evolved FSMs for a given k (shaded diagonal). These algorithms were also used for a robustness check using a different number of k (non diagonal elements). The bottom row d_M denotes the maximal initial Manhattan distance of all agents to their target, averaged over all 100 configurations of one set.

	Set k=1		Set k=2		Set k=4		Set k=8		Set k=16		Set k=32		Set k=64		Set k=128		Set k=256	
	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps	Succ.	Time Steps
Top2	100	23.94	100	30.03	100	35.38	100	40.72	100	44.57	100	50.15	99	54.72	86	61.42	18	76
k=1	100	24.01	100	30.2	100	35.78	99	40.65	99	44.39	99	49.44	98	53.63	92	59.71	42	74.21
Top2	100	24.35	100	30.33	100	35.49	100	40.75	100	44.26	100	50.03	100	53.86	95	59.97	39	73.67
k=2	100	24.6	100	30.63	100	35.83	100	41.32	100	45.07	100	50.07	99	53.77	95	60.34	35	73.57
Top2	100	23.99	100	30.1	100	35.36	100	40.71	100	44.31	100	50.04	99	53.34	94	60.14	39	73.31
k=4	100	24.05	100	30.04	100	35.38	100	40.69	100	44.34	100	49.6	98	53.54	96	59.46	41	72.15
Top2	100	24.53	100	30.2	100	36.14	100	40.39	100	44.39	98	49.8	96	54.11	82	60.54	38	74.45
k=8	100	24.11	100	30.11	100	35.79	100	40.39	100	44.57	100	49.32	100	53.42	96	59.92	45	72.98
Top2	100	23.99	100	30.15	100	35.75	100	40.6	100	44.4	100	49.71	99	53.96	94	59.78	45	71.93
k=16	100	24.15	100	30.45	100	35.93	100	40.57	100	44.43	100	49.43	100	53.39	92	60	46	73.41
Top2	100	24.27	100	30.18	100	35.79	100	40.46	100	44.61	100	49.06	99	53.43	94	59.98	43	74.53
k=32	100	24.33	100	30.26	100	35.89	100	40.78	100	44.55	100	49.2	100	53.54	95	59.8	42	74.5
Top2	100	24.2	100	30.09	100	35.42	100	40.62	100	44.68	100	49.47	100	53.18	95	59.77	52	75.48
k=64	100	23.94	100	30.05	100	35.29	100	40.68	100	44.25	100	49.68	100	53.4	96	59.67	50	74.74
Top2	100	26.38	100	32.46	100	38.72	100	43.39	100	47.05	100	51.86	99	56.42	100	60.94	55	73.49
k=128	100	26.31	100	32.08	100	38.82	100	42.85	100	46.88	100	51.83	99	57.05	99	62.09	42	76.17
Top2	100	42.1	100	52.94	100	63.24	100	72.25	98	80.36	100	87.98	98	93.17	99	100.9	100	115
k=256	100	46.45	100	60.3	100	70.37	100	76.99	100	87.14	100	94.2	100	100.8	100	109.3	100	124.4
max(d_M)		22.15		28.01		33.48		38.18		41.66		45.94		48.47		50.39		52.55

4.3 Directed agents (M3)

The Top 1 algorithms are successful for $k = 1 \dots 256$ (Table 2, Fig. 7). The Top 10 algorithms’ performance is listed in Table 3 for $k = 128, 256$. Not all of the Top 10 ($k = 128$) are completely successful. One reason might be that for $k = 128$ better algorithms might exist that could be evolved through a longer running genetic algorithm.

Table 3
The Top 10 algorithms’ performance for the $k = 128$ and $k = 256$ evolved directed agents, evaluated for 100 initial configurations of the Ranking Set with $k = 128, 256$ agents. The Top 10 ($k = 256$) algorithms are also good for $k = 128$, but the Top 10 ($k = 128$) algorithms are weak for $k = 256$ agents. The column “Targets Missed” denotes the mean number of targets that could not be reached when an algorithm was not successful.

	Set k= 128			Set k= 256		
	Succ.	Targets Missed	Time Steps	Succ.	Targets Missed	Time Steps
Top10 k=128	100	0	60.94	55	19.33	73.49
	99	4	62.09	42	25.26	76.17
	99	5	61.17	42	27.43	73.98
	99	6	61.43	42	22.09	75.02
	99	6	62.27	49	23.10	76.57
	99	14	61.07	66	20.00	74.48
	99	17	62.74	41	21.51	77.07
	98	6	62.58	40	23.65	75.93
	98	7	61.38	64	21.56	74.59
	98	7.5	61.38	43	25.42	74.72
Top10 k=256	99	4	100.9	100	0	115
	100	0	109.3	100	0	124.4
	99	6	115.5	100	0	142.6
	100	0	123.2	100	0	145.6
	100	0	145.9	100	0	168.6
	100	0	140.8	100	0	172.2
	100	0	147.6	100	0	173.6
	100	0	139.3	100	0	176.1
	100	0	149.7	100	0	177.8
	100	0	143	100	0	178.5
max(d _M)	50.39			52.55		

4.4 Randomized directed agents (M4)

The randomized directed agent is completely successful for $k = 1 \dots 16$ and almost as fast as the agent M3 (Fig. 7). For $k = 32, 64, 128, 256$ the success rate is decreasing (99.99%, 99.96%, 98.4%, 71.6%). As for the undirected agents, the congestion problem becomes more relevant with a higher number of agents leading to an increase of time steps needed to find all the targets. But because of the readable direction and the swapping technique which both can avoid conflicts in spite of spatial closeness, the directed agents have an advantage.

4.5 Robustness check

The Top 10 (k) algorithms were simulated on all the other initial configurations which were defined for another number k' of agents in the Ranking Sets. This

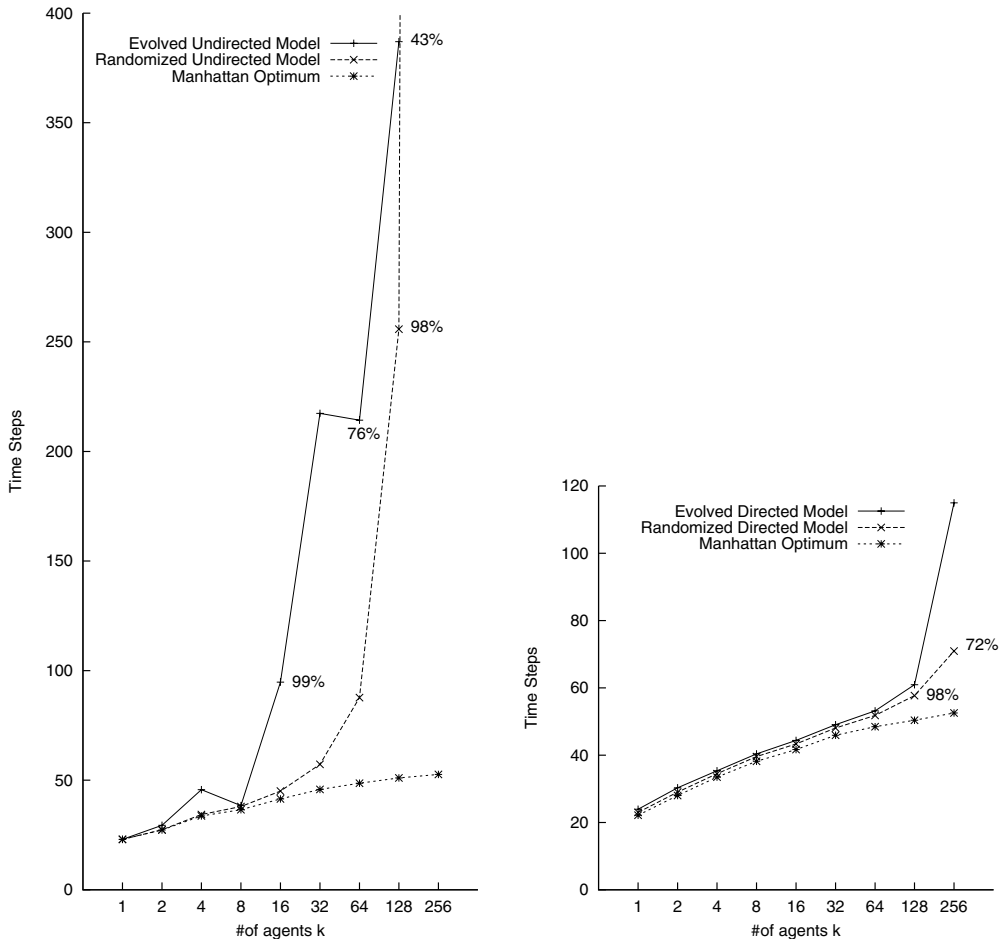


Fig. 7. Number of generations to solve the task for undirected agents (left graph) and for directed agents (right graph). If a percentage is noted then only $p\%$ of the 100 initial configurations were solved, otherwise all.

test was performed in order to check how robust an algorithm (that was evolved specifically for k agents) is against changes of k . The results are given in Table 1 and Table 2. The values in the diagonal (shaded) are for the evolved algorithms for a certain k . The other values in a line show the performance of that algorithm for another $k' < k$ (left from diagonal) or $k' > k$ (right from diagonal). The general observation is that algorithms which are evolved for a certain k can also be useful for a lower k' but not so for a higher k' .

Further tests have been performed with manually designed environments for the model M3, in which $k = 256$ agents were placed with a random initial direction in 32×32 environments with border. The best evolved and ranked algorithms were simulated on them. Fig. 8 shows a simulation sequence on such an environment. The movement strategies are different but successful in both cases. Since the agents' targets were placed on the opposite site (both in x-direction and in y-direction) than the agents starting position, it was expected that the agents will use the way through the middle of the environment like in Fig. 8 (b). But it seems to be an even better

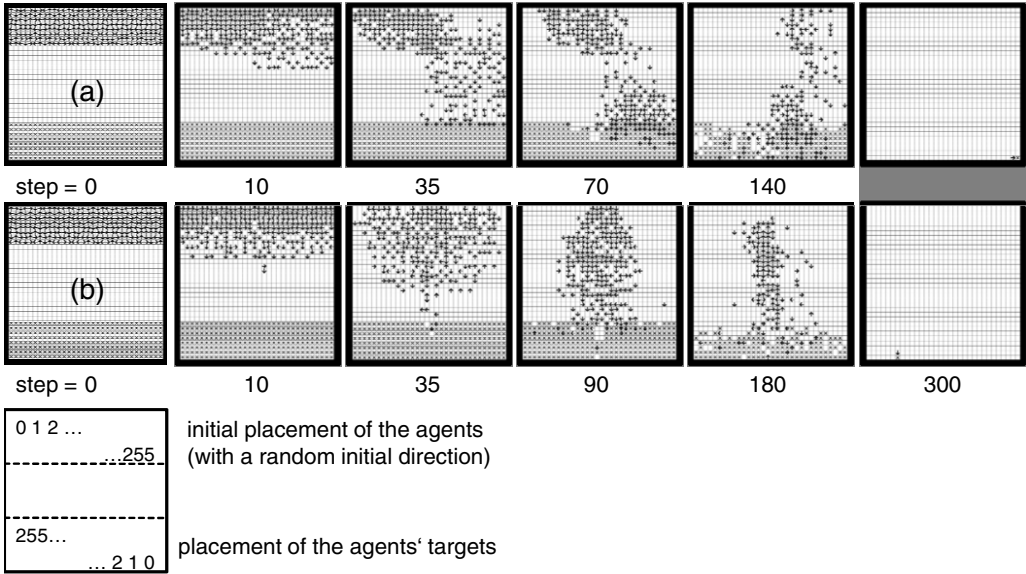


Fig. 8. Simulation Sequence of a manually generated environment. The best algorithm evolved for $k = 256$ (a) needs 228 steps, the third-best algorithm evolved for $k = 256$ (b) needs 301 steps.

strategy to walk clockwise around the center to avoid congestion (Fig. 8 (a)).

5 Conclusion

Four different CA compliant models were proposed to model the multiple target search problem with agents: (1) undirected agents, (2) random undirected agents, (3) directed agents, (4) random directed agents. The moving and turning actions of an undirected and directed agent are controlled by a MEALY control finite state machine (also called control algorithm) that is a part of the whole cell automaton. Control algorithms for a different number of agents ($k = 1 \dots 256$) were evolved using an island model genetic algorithm for a 32×32 environment with border.

The most reliable (with respect to solve the problem for all initial configurations) algorithms were found for the M3 model (directed agent). They are completely successful for the 100 random initial configurations of the ranking set for up to 256 agents. The performance (speed, number of generation needed) is very good for up to 128 agents and good for 256 agents. As we cannot test all possible initial configurations, we cannot be sure that no deadlocks may occur. A formal proof seems to be very difficult, too, because of the overwhelming amount of possible situations regarding the number of involved agents, their current state and position relatively to each other. The algorithms for the M1 model (undirected agents) are the weakest with respect to reliability and speed. A random undirected agent behaves better than a controlled undirected agent. A random directed agent is faster than a controlled directed agent, but not reliable for $k = 128, 256$.

Future work

The deterministic models M1 and M3 can be improved using more inputs (conflicts, obstacle, swap) and different input reduction tables for the control automaton. The number of control states can be increased, too. A time-shuffling technique (changing the algorithm in time [1]) could be used (use dynamically another algorithm when the number of agents is decreasing). The randomized models M2 and M4 could be improved in order to escape from deadlocks. The problem can be generalized by allowing obstacles, moving targets or explicit communication between the agents.

References

- [1] Ediger, P. and R. Hoffmann, *Improving the behavior of creatures by time-shuffling*, in: H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki and S. Bandini, editors, *ACRI*, Lecture Notes in Computer Science **5191** (2008), pp. 345–353.
- [2] Ediger, P. and R. Hoffmann, *Optimizing the creature’s rule for all-to-all communication*, in: *EPSRC Workshop Automata-2008. Theory and Applications of Cellular Automata, Bristol, UK, June 12-14, 2008.*, 2008, pp. 398–410.
- [3] Ediger, P., R. Hoffmann and M. Halbach, *Evolving 6-state automata for optimal behaviors of creatures compared to exhaustive search*, in: R. Moreno-Díaz, F. Pichler and A. Quesada-Arencibia, editors, *EUROCAST*, Lecture Notes in Computer Science (2009).
- [4] Goldenberg, M., A. Kovarsky, X. Wu and J. Schaeffer, *Multiple agents moving target search*, in: G. Gottlob and T. Walsh, editors, *IJCAI* (2003), pp. 1536–1538.
- [5] Halbach, M., “Algorithmen und Hardwarearchitekturen zur optimierten Aufzählung von Automaten und deren Einsatz bei der Simulation künstlicher Kreaturen,” Ph.D. thesis, Technische Universität Darmstadt (2008).
- [6] Halbach, M., R. Hoffmann and L. Both, *Optimal 6-state algorithms for the behavior of several moving creatures*, in: S. El Yacoubi, B. Chopard and S. Bandini, editors, *ACRI*, Lecture Notes in Computer Science **4173** (2006), pp. 571–581.
- [7] Hochberger, C., R. Hoffmann and S. Waldschmidt, *CDL++ for the description of moving objects in cellular automata*, in: V. E. Malyskin, editor, *Parallel Computing Technologies, 5th International Conference, PaCT-99, St. Petersburg, Russia, September 6-10, 1999, Proceedings*, Lecture Notes in Computer Science (LNCS) **1662**, Springer-Verlag (Berlin), St. Petersburg, Russia, 1999 pp. 428–435.
- [8] Komann, M., P. Ediger, D. Fey and R. Hoffmann, *On the effectivity of genetic programming compared to the time-consuming full search of optimal 6-state automata*, in: L. Vanneschi, S. Gustafson and M. Ebner, editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, LNCS (2009).
- [9] Komann, M., A. Mainka and D. Fey, *Comparison of evolving uniform, non-uniform cellular automaton, and genetic programming for centroid detection with hardware agents*, in: V. E. Malyskin, editor, *PaCT*, Lecture Notes in Computer Science **4671** (2007), pp. 432–441.
- [10] Korf, R. E., *Real-time heuristic search*, Artificial Intelligence **42** (1990), pp. 189–211.
- [11] Loh, P. K. K. and E. C., *Performance simulations of moving target search algorithms*, International Journal of Computer Games Technology **2009** (2009), pp. 1–6.
- [12] Schadschneider, A., *Conflicts and friction in pedestrian dynamics*, in: H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki and S. Bandini, editors, *Cellular Automata, 8th International Conference on Cellular Automata for Research and Industry, ACRI 2008, Yokohama, Japan, September 23-26, 2008. Proceedings*, Lecture Notes in Computer Science **5191** (2008), pp. 559–562.
- [13] Schadschneider, A. and M. Schreckenberg, *Cellular automaton models and traffic flow*, Journal of Physics A **26** (1993), p. L679.
- [14] Sipper, M., “Evolution of Parallel Cellular Machines, The Cellular Programming Approach,” Lecture Notes in Computer Science **1194**, Springer, 1997.
- [15] Sipper, M. and M. Tomassini, *Computation in artificially evolved, non-uniform cellular automata*, Theor. Comput. Sci. **217** (1999), pp. 81–98.

- [16] Spicher, A., N. Fatès and O. Simonin, *From reactive multi-agent models to cellular automata - illustration on a diffusion-limited aggregation model*, in: *1st International Conference on Agents and Artificial Intelligence, Portugal (2009)*, 2009, pp. 422–429.
- [17] Tavakoli, Y., H. H. S. Javadi and S. Adabi, *A cellular automata based algorithm for path planning in multi-agent systems with a common goal*, IJCSNS, International Journal of Computer Science and Network Security **8** (2008), pp. 119–123.
- [18] Toffoli, T. and N. Margolus, “Cellular Automata Machines: A New Environment for Modeling,” MIT Press, Cambridge, MA, USA, 1987.