Evolving Multiplication as Emergent Behavior in Cellular Automata Using Conditionally Matching Rules

Michal Bidlo

Abstract— **In this paper a special representation technique called conditionally matching rules will be applied in order to design computational processes in uniform cellular automata. The goal is to verify abilities of this approach in combination with genetic algorithm on the problem of disigning various cellular automata that exhibit a given computational process. The principle of a computational process in a cellular automaton is to interpret some cells as input bits and some (possibly other) cells as output bits (i.e. the result of the computation). The genetic algorithm is applied to find a suitable transition function of a cellular automaton according to which the given computation could be observed during its development for all the possible binary combinations stored in the input cells. Both the input values and the result is represented by state values of cells. The input of the computation will be represented by the initial state of the cellular automaton. After a finite number of development steps the cells representing the output bits are expected to contain the result of the computation. A set of experiments will be performed considering various setups of the evolutionary system and arrangements of the target computation. It will be shown that non-trivial computations can be realized in a uniform two-dimensional cellular array.**

I. Introduction

Cellular automata (CA) represent a biologically inspired dynamical system with a discrete time and space. Cells represent basic computational elements of a CA. At a given moment each cell possesses a value representing its state from a finite set of states. The cell states can be considered as data (information) units processed by the cellular automaton. The concept of cellular automata was originally invented by Ulam and von Neumann in 1966 [1] for studying the behavior of complex systems.

A two-dimensional (2D) cellular automaton consists of a regular grid of cells that are arranged into a regular matrix (mesh). In each (discrete) developmental step of the CA the states of all the cell are updated synchronously in parallel according to a local transition function. The next state of a given cell depends on the combination of states in its neighborhood (including the cell itself). A sequence of updating the cell states during discrete time steps represents development of the cellular automaton.

For the purposes of this paper the following concept of the cellular automata will be considered. The cellular neighborhood of each cell is represented by a 9-tuple (3x3 cells) consisting of the investigated (central) cell and its immediate neighbors in the horizontal, vertical and diagonal directions. Since only finite-size cellular automata can be practically implemented, boundary conditions will be defined for the cells at the border of the cellular mesh. In this paper zero boundary conditions will be applied which means that non-existing neighbors of the border cells are considered as notional cells in a permanent state 0.

Conventionally the local transition function is represented by a table that specifies the next state of a cell for all the possible combinations of states in its neighborhood. However, if the number of cell states or the size of the cellular neighborhood increases, then the number of such combinations grows exponentially and thus the representation and design of the transition function becomes a challenging task.

In order to overcome this issue, a new technique for representing the transition functions was introduced by Bidlo et al. and called as Conditionally Matching Rules (CMR) [2]. This approach is fundamentally inspired by the conventional table-based representation. It means that the CMR encoding allows to specify the transition rules as usual in the table-based approach but, in addition to that, more general rules can be formulated whose interpretation covers several common rules in a single CMR. In particular, a conditionally matching rule consists of a conditional part and a next state. The conditional part encodes a series of pairs — a condition function and a state value — whose number corresponds to the number of cells in the cellular neighborhood. The structure of a CMR is illustrated in the upper part of Figure 1. A local transition function of a CA consists of a finite sequence of conditionally matching rules. The process of determining the next state of a cell using the CMR-based transition function is the following. The CMRs are evaluated sequentially one after another until a CMR matches the states in the cellular neighborhood. In order to determine a CMR match, each rule of its conditional part is evaluated with respect to the corresponding cell state in the neighborhood (see Figure 1). If all the conditions are satisfied, then the next state from the matching CMR represents the result of the transition function (i.e. the new state of the investigated cell) and none of the remaining CMRs in the sequence needs to be evaluated. If none of the CMRs representing the transition function matches, then the cell keeps its current state. The experiments showed that if the CMR approach is utilized to represent transition functions, then more complex cellular automata can be effectively evolved in comparison with the traditional representation [2]. Hence the advanced features and abilities of the CMR representation are worth the next investigation.

In addition to a wide range of applications utilizing cellular automata to solve some specific tasks (e.g. modeling complex biological and physical systems, artificial live, random number generation and many others [3]), cellular automata

Michal Bidlo is with the Faculty of Information Technology, Brno University of Technology, IT4Innovations Centre of Excellence, Božetěchova 2, 61266 Brno, Czech Republic, email: bidlom@fit.vutbr.cz.

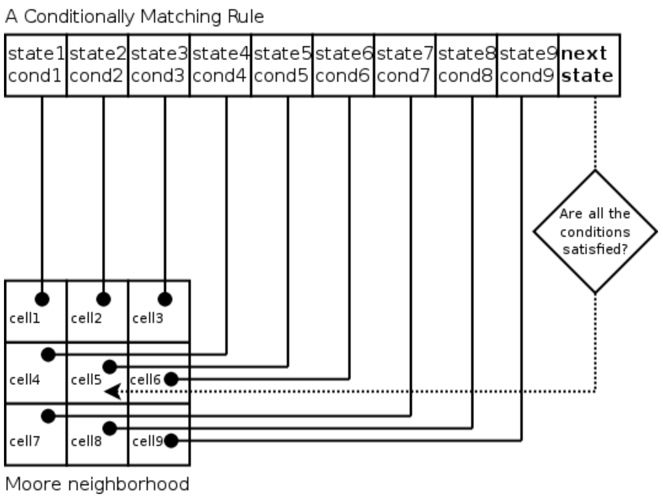


Fig. 1. Structure and interpretation of a conditionally matching rule

also represent a platform to perform computations. Various concepts of computation performed in cellular automata have been studied both theoretically and practically (i.e. using real implementations in FPGAs or as application-specific circuits [4][5]). The importance of undertaking such kind of research is motivated especially by the fact that cellular automata represent (in many cases) homogeneous and massive parallel computing platform with typically local interactions of cells. The issue of homogeneity may be important in a process of designing large systems whose elements (computational units – cells), for example, perform a given function that, in cooperation with each other, realizes a specific (emergent) behavior. An advantage of such system may be its scalability (it is easy to connect additional cells with the same function) or a possibility of repair in case of a cell failure. Advanced concepts may consider specializations of different cells during the system functioning. This idea is mostly inspired by multicellular (biological) systems in which a target organism can grow during its life and the cells change their kind with respect to a location in the organism or on the basis of external conditions. Some studies and applications of these issues (in a more general conception referred as computation development) were published in [6]. Computational universality of cellular automata was proven for the first time by their inventor John von Neumann in [1]. His CA worked with 29 cell states which was later reduced to 8 states by Codd [7]. Lindgren and Nordahl demonstrated that even 1D cellular automaton can be utilized as universal computing platform (i.e. a platform that is able to simulate the computation using Turing machine). They proposed a proof of universality for the 7-state 1D CA with 3-cell neighborhood and 4-state 1D CA with 5-cell neighborhood [8]. Sipper showed that 2D binary non-uniform cellular automata are able to perform computations by demonstrating how to realize computationally complete set of logic functions and their interconnection [4]. Uniform 2D binary cellular automaton was demonstrated to be a computationally universal platform using the popular Conway's Game of Life rules [9]. The idea was to utilize some "living" cell structures like glider guns, gliders, periodic patterns etc. to simulate the computational process of Turing machine. These structures represent a means for implementing basic logic gates, synchronization mechanism and memory elements which represent fundamental components of a universal computer. Another interesting computationally universal 2D CA is represented by the rules of Langton's ant [10]. His CA represents a model with simple transition rules that is able to exhibit complex emergent behavior. A proof of universality of Langton's ant was proposed in [11]. Other (not necessarily universal) cellular automata able to perform specific compatational tasks were also published (e.g. Langton's loop implementing self-replication [12], Tempesti loop which added an ability of construction [13] and others).

The goal of this paper is to show that various CA setups together with some appropriate evolutionary system setups are able to design transition functions allowing us to perform a specific computational task in the CA. In particular, we propose two sets of condition functions for the CMR in combination with various input and output cell arrangements in order to design CA whose development exhibits 2x2-bit multiplication. The objective of studying various CMR setups is to determine how the different sets of condition functions influence the ability of the evolutionary algorithm to find a solution of a given task in cellular automata. The experiments will be evaluated with respect to the success rate and computational effort of the evolutionary design process. Features and abilities of the obtained results will be discussed with respect to a future research.

II. Setup of Conditionally Matching Rule

The original concept of the conditionally matching rules introduced in [2] considered ordinary relational operators equal to ($==$), not equal to ($!=$), greater or equal than ($>=$), less or equal than ($<=$) and a don't care mask ($\star$) as the set of conditions. In case of binary cellular automata, where the state of a cell can be either 0 or 1, each condition in a CMR actually represent a function with two binary inputs (where one bit represents the state of a cell from the cellular neighborhood, the other is a state value specified in the conditional part). The function calculates a single-bit output indicating whether the given part of the CMR matches the corresponding cell state. Table I shows results of the aforementioned functions for all their possible input values.

Since there are 4 possible input combinations, $2^4 = 16$ different functions exist in total which could be used as condition functions in CMRs. Our previous experiments showed that it is not suitable to consider all the 16 functions because the space of the transition functions becomes very huge and the problem of finding a specific behavior of the CA represents a challenging task. Therefore, the selection of a subset of condition functions represents a reasonable solution. However, the main question is how to select this subset for the CMR in order to allow efficient design of transition functions for cellular automata. In [2] it was demonstrated that the subset of functions summarized in

TABLE I

THE TRUTH TABLE FOR THE ORIGINAL CONDITION FUNCTIONS $==$, $!=$, $>=$, $<=$ AND A DON'T CARE $\star$. IF THE RESULT OF A CONDITION FUNCTION IS 1, THEN THE APPROPRIATE CELL STATE MATCHES THE STATE SPECIFIED IN THE CONDITIONAL PART WITH RESPECT TO THIS FUNCTION. S_{CELL} DENOTES THE STATE OF A CELL IN THE CELLULAR NEIGHBORHOOD, S_{CMR} REPRESENTS THE STATE SPECIFIED WITHIN A CONDITION IN THE CMR.

Inputs to condition function		**Condition function**				
S_{CELL}	S_{CMR}	$==$	$!=$	$\leq$	$\geq$	$\star$
0	0	1	0	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	1	1	1

Table I represents one of the possible choices in order to achieve non-trivial behavior in binary 2D cellular automata. This subset was chosen experimentally on the basis of analyzing the target CA behavior (the replication and pattern transformation task).

In order to determine whether a more suitable subset of condition functions exists, the following approach was considered for the experiments presented in this paper. There are in total $2^{16} = 65536$ different subsets considering the complete set of 16 condition functions. It is possible to perform exhaustive search through all the subsets and evaluate the number of different functions that can be realized using the proposed CMR-based representation. For the purposes of this paper, the problem of calculating single-output binary functions with 9 inputs was considered in order to evaluate each subset of the condition functions. In fact, this setup is identical to exploring transition functions for 2D uniform binary cellular automata with 9-cell Moore neighborhood. Since it is impossible to perform in a reasonable time the exhaustive search of all tle possible 9-input functions, the evaluation was performed by generating 1 billion random samples of CMR sequences (considering the number of CMRs from 1 to 8). This experiment showed that one of the highest numbers of different functions can be generated using the subset of condition functions summarized in Table II. As evident, the resulting subset of condition functions contains, in addition to the relational operators $<=$ and $>=$, the identity function of the cell state (let us denote it as $id(S_{CELL})$ where S_{CELL} represents the cell state), the negation of S_{CELL} ($not(S_{CELL})$) and does not contain don't care mask ($\star$). Note that in this experiment no specific function (or CA behavior) was required, only the number of various functions was observed. The goal of this experiment was to identify a subset of condition functions for the CMR representation that would be potentially able to solve as wide set of tasks as possible.

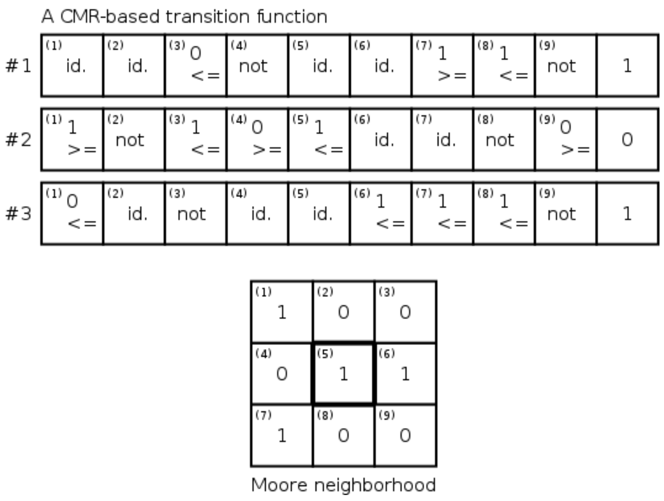
Figure 2 shows an example of transition function represented by three conditionally matching rules. The CMRs utilize the condition functions from Table II. In order to determine the next state of the investigated cell (denoted by the thick rectangle in Figure 2), the CMRs are evaluated

TABLE II

THE TRUTH TABLE FOR A NEW SET OF CONDITION FUNCTIONS THAT MAY POTENTIALLY BE ABLE TO GENERATE A WIDER RANGE OF FUNCTIONS USING THE CMR REPRESENTATION.

Inputs to condition function		**Condition function**			
S_{CELL}	S_{CMR}	$id(S_{CELL})$	$not(S_{CELL})$	$\leq$	$\geq$
0	0	0	1	1	1
0	1	0	1	1	0
1	0	1	0	0	1
1	1	1	0	1	1

sequentially one after another in order to find a CMR that matches the state of the cellular neighborhood. For the CMR #1 it can be seen that condition (2) — the identity function — does not match because the state of cell (2) in the neighborhood has state 0. Therefore, CMR #1 can not be applied to determine the next state. Considering the CMR #2, it can be verified that this CMR fulfills all its conditions with respect to the states in the cellular neighborhood (the rule (1) satisfies the condition $S_{CELL} >= 1$ because $S_{CELL(1)} = 1$, the rule (2) after its evaluation matches the state of cell (2) because $not(S_{CELL(2)}) = not(0) = 1$ and so on). Therefore, the CMR #2 will be applied to determine the next state of the investigated cell, i.e. its new state will be 0.



A CMR-based transition function
#1 (1) id. (2) id. (3) 0 <= (4) not (5) id. (6) id. (7) 1 >= (8) 1 <= (9) not 1
#2 (1) 1 >= (2) not (3) 1 <= (4) 0 >= (5) 1 <= (6) id. (7) id. (8) not (9) 0 >= 0
#3 (1) 0 <= (2) id. (3) not (4) id. (5) id. (6) 1 <= (7) 1 <= (8) 1 <= (9) not 1
(1) 1 (2) 0 (3) 0
(4) 0 (5) 1 (6) 1
(7) 1 (8) 0 (9) 0
Moore neighborhood

Fig. 2. Example of a transition function represented by conditionally matching rules. Note that the identity function (id) and negation (not) do not need any state value in the conditional part of the CMR because the decision of matching their part of CMR is based only on evaluating the appropriate cell state in the cellular neighborhood. The thick rectangle denotes the cell for which the new state ought to be calculated.

III. EVOLUTIONARY SYSTEM SETUP

Genetic algorithm (GA) was utilized for the evolution of CMR-based transition functions in order to realize 2x2-bit multiplication in 2D uniform binary cellular automata.

The population consists of 16 individuals (chromosomes) that are initialized randomly at the beginning of the evolutionary process. Each chromosome represents a candidate transition function represented as a finite sequence of CMRs.

The structure of each CMR is identical to that shown in the top part of Figure 1. Each CMR is encoded as a finite sequence of integers representing the conditional parts (i.e. the states and condition functions) and the next state.

The fitness function implements the following multiplication scheme in the cellular automata for evaluating the chromosomes. A binary input test vector (representing the operands to be multiplied) is generated into the given cells (let's call them the input cells) as the initial state of the CA. For 2x2-bit multiplication there are 4 input cells, i.e. two 2-bit operands. All the other cells are initialized by the state 0. Now the CA performs 16 development steps according to the transition function encoded in the chromosome after which the result of multiplication is verified as a sequence of states of the given (output) cells. For 2x2-bit multiplication there are 4 output cells, i.e. a 4-bit product. The fitness value is increased by one for every correct bit of the result with respect to the input vector. The evaluation is performed for all the possible input test vectors. For 2x2-bit multiplication there are $n = 2^4 = 16$ input test vectors and $p = 4$ bits of the product. Therefore, the maximal fitness for the 2x2-bit multiplication $F_{mult} = n \times p = 16 \times 4 = 64$. Moreover, we required the cells of the result to keep the states representing the product during the subsequent CA development. Hence the cell states representing the product are compared to the values after performing one more step of the CA for each of the input test vectors. If all the output cells keep their resulting states, the fitness is increased by one. Note that the input values are usually modified during the CA development (there is no requirement to keep them within the input cells). The maximal value of the complete fitness can be expressed as $F_{max} = F_{mult} + n = 64 + 16 = 80$.

Each step of the evolution is performed by generating offspring from parent chromosomes using a mutation operator until an entire new population is created. The parent chromosome is selected using tournament operator out of T chromosomes randomly chosen from the actual population, i.e. the fittest individual from the group of T chromosomes becomes the parent. The parameter T is referred to as the base of the tournament selection operator. The parent undergoes mutation as follows. A random integer M in the range from 1 to 4 is generated. Then M random positions within the parent chromosome are selected. The offspring is created by replacing the actual genes at these positions by new randomly generated values. No crossover operator is applied.

In order to explore the possibilities of realizing the proposed multiplication scheme, several sets of experiments were performed considering various setups. The two sets of condition functions presented in Table I and II were investigated within the evolutionary process. The GA was parametrized by the base of the tournament operator for $T = 2, 4, 6, 8$. The number of CMRs ($\#CMRs$) of the transition function was considered for $\#CMRs = 6, 8, 10, 12$. For each combination of these parameters, several setups of the input and output cells were considered as shown in Figure

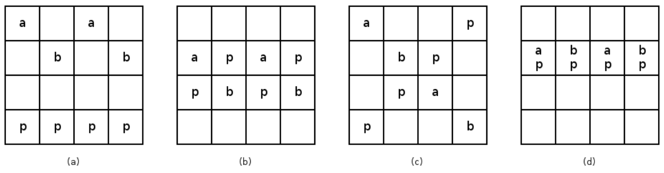


Fig. 3. The setups for input and output cell arrangements in cellular automata: (a) separated, (b) alternating, (c) diagonal, (d) shared. In each of the setups, a and b denote the input cells whose states represent values for the 2-bit operands, p denote the output cells in which the resulting product is expected after the CA development. Note that in the shared setup (d) the result is expected in the same cells as the input operands.

3. The cellular automaton consisting of 10x10 cells was used for the evaluation of the chromosomes. The structure containing the input and output cells is located in the middle of the CA. The reason for choosing larger CA is not to strictly limit the computation process by the boundary conditions (required for finite-size CAs).

For each set of experiments (specified by the aforementioned parameters and cell setups) 100 independent evolutionary runs were performed. The evolution is terminated if the desired behavior of the candidate CA is observed (i.e. a chromosome with the maximal fitness value is found) or if a limit of 100 thousands generations is reached.

IV. Experimental Results

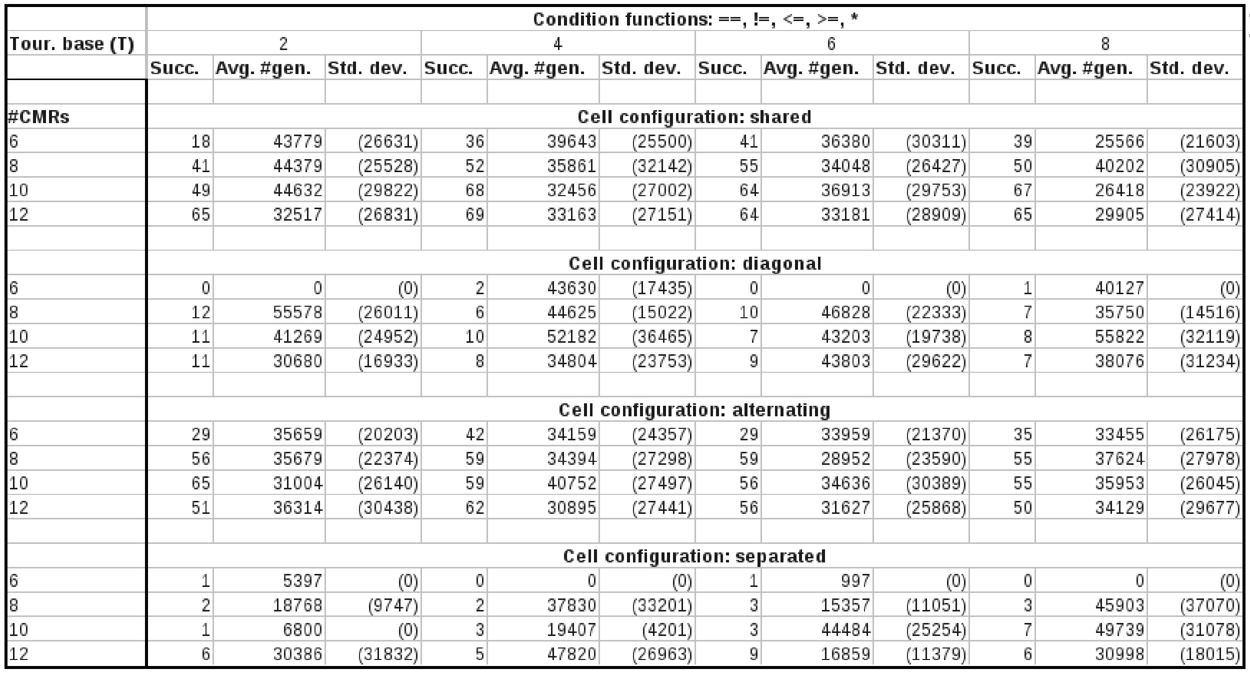
In this section we propose an overview of the experimental results obtained from the evolutionary system described in Section III. For each set of experiments the success rate and computational effort (measured as the number of generations of the GA needed to find a solution) were evaluated. The experiments showed that it is possible to realize the given computational task in uniform 2D binary cellular automata. We also determined that the required behavior discovered by the evolution may not be limited to a specific CA size.

The results of evolution using the original condition functions from Table I are shown in Table IIIa. The results of the application of the new set of condition functions from Table II are summarized in TableIIIb. In each set of experiments the success rate and computational effort of the evolutionary process was investigated with respect to the base of tournament selection (T) and the number of conditionally matching rules ($\#CMRs$) of the transition function. As the results show for both sets of the condition functions, the success rate tends to increase with increasing the $\#CMRs$ which indicates that although a larger search space is needed to explore (with potentially higher amount of target solutions), then the evolution is able to explore it effectively and in many cases even with less computational effort (expressed by the number of generations $Avg.\#gen$). Similar trend can also be observed for the increasing the base of tournament selection T. This parameter actually increases the selection pressure during evolution (the higher the T the higher the selection pressure) which means that the individuals with higher fitness are able to generate offspring chromosomes towards the target solutions within the search

TABLE III

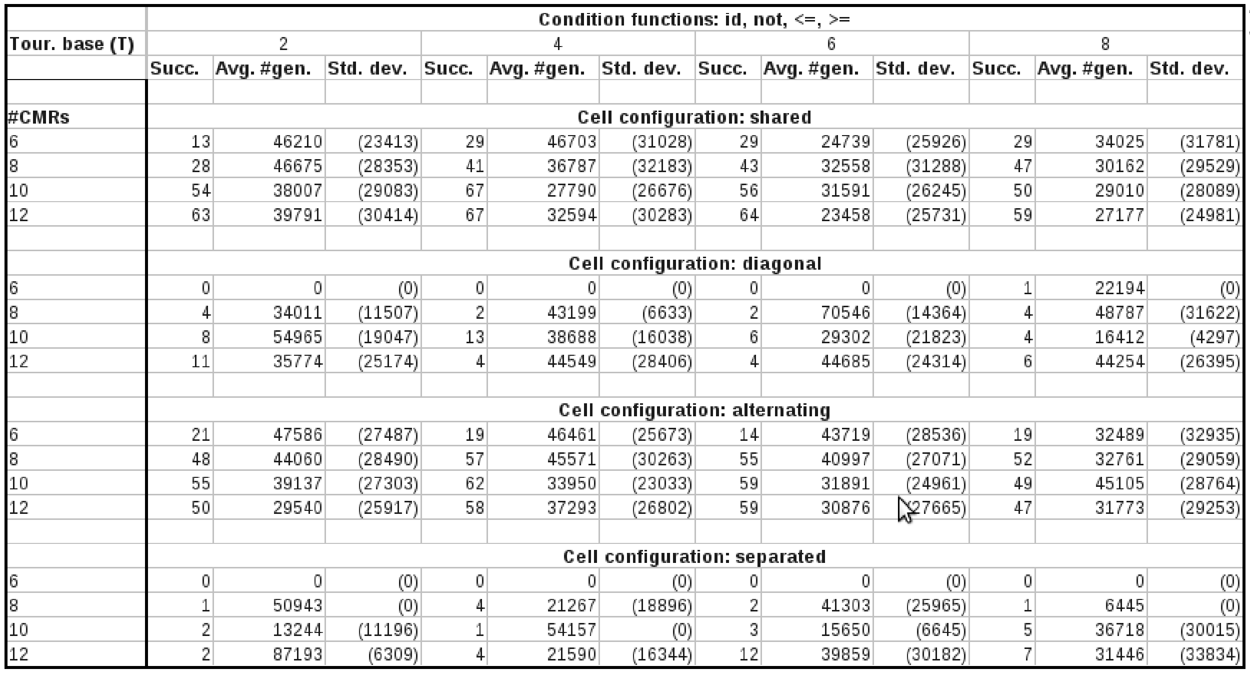
Statistical results of the evolutionary experiments performed for various numbers of conditionally matching rules ($\#CMRs$) and various base values of the tournament selection in GA (T). The success rate (**Succ.**) and average number of generations (**Avg.#gen**) needed to find a working solution were measured. **Std. dev.** denote the standard deviation calculated from the number of generations of the successful evolutionary runs.

(a)



	Condition functions: ==, !=, <=, >=, *											
Tour. base (T)	2			4			6			8		
	Succ.	Avg. #gen.	Std. dev.	Succ.	Avg. #gen.	Std. dev.	Succ.	Avg. #gen.	Std. dev.	Succ.	Avg. #gen.	Std. dev.
#CMRs	Cell configuration: shared											
6	18	43779	(26631)	36	39643	(25500)	41	36380	(30311)	39	25566	(21603)
8	41	44379	(25528)	52	35861	(32142)	55	34048	(26427)	50	40202	(30905)
10	49	44632	(29822)	68	32456	(27002)	64	36913	(29753)	67	26418	(23922)
12	65	32517	(26831)	69	33163	(27151)	64	33181	(28909)	65	29905	(27414)
	Cell configuration: diagonal											
6	0	0	(0)	2	43630	(17435)	0	0	(0)	1	40127	(0)
8	12	55578	(26011)	6	44625	(15022)	10	46828	(22333)	7	35750	(14516)
10	11	41269	(24952)	10	52182	(36465)	7	43203	(19738)	8	55822	(32119)
12	11	30680	(16933)	8	34804	(23753)	9	43803	(29622)	7	38076	(31234)
	Cell configuration: alternating											
6	29	35659	(20203)	42	34159	(24357)	29	33959	(21370)	35	33455	(26175)
8	56	35679	(22374)	59	34394	(27298)	59	28952	(23590)	55	37624	(27978)
10	65	31004	(26140)	59	40752	(27497)	56	34636	(30389)	55	35953	(26045)
12	51	36314	(30438)	62	30895	(27441)	56	31627	(25868)	50	34129	(29677)
	Cell configuration: separated											
6	1	5397	(0)	0	0	(0)	1	997	(0)	0	0	(0)
8	2	18768	(9747)	2	37830	(33201)	3	15357	(11051)	3	45903	(37070)
10	1	6800	(0)	3	19407	(4201)	3	44484	(25254)	7	49739	(31078)
12	6	30386	(31832)	5	47820	(26963)	9	16859	(11379)	6	30998	(18015)

(b)



	Condition functions: id, not, <=, >=											
Tour. base (T)	2			4			6			8		
	Succ.	Avg. #gen.	Std. dev.	Succ.	Avg. #gen.	Std. dev.	Succ.	Avg. #gen.	Std. dev.	Succ.	Avg. #gen.	Std. dev.
#CMRs	Cell configuration: shared											
6	13	46210	(23413)	29	46703	(31028)	29	24739	(25926)	29	34025	(31781)
8	28	46675	(28353)	41	36787	(32183)	43	32558	(31288)	47	30162	(29529)
10	54	38007	(29083)	67	27790	(26676)	56	31591	(26245)	50	29010	(28089)
12	63	39791	(30414)	67	32594	(30283)	64	23458	(25731)	59	27177	(24981)
	Cell configuration: diagonal											
6	0	0	(0)	0	0	(0)	0	0	(0)	1	22194	(0)
8	4	34011	(11507)	2	43199	(6633)	2	70546	(14364)	4	48787	(31622)
10	8	54965	(19047)	13	38688	(16038)	6	29302	(21823)	4	16412	(4297)
12	11	35774	(25174)	4	44549	(28406)	4	44685	(24314)	6	44254	(26395)
	Cell configuration: alternating											
6	21	47586	(27487)	19	46461	(25673)	14	43719	(28536)	19	32489	(32935)
8	48	44060	(28490)	57	45571	(30263)	55	40997	(27071)	52	32761	(29059)
10	55	39137	(27303)	62	33950	(23033)	59	31891	(24961)	49	45105	(28764)
12	50	29540	(25917)	58	37293	(26802)	59	30876	(27665)	47	31773	(29253)
	Cell configuration: separated											
6	0	0	(0)	0	0	(0)	0	0	(0)	0	0	(0)
8	1	50943	(0)	4	21267	(18896)	2	41303	(25965)	1	6445	(0)
10	2	13244	(11196)	1	54157	(0)	3	15650	(6645)	5	36718	(30015)
12	2	87193	(6309)	4	21590	(16344)	12	39859	(30182)	7	31446	(33834)

space. However, the comparison of the results in Table III shows that in most cases the success rate is lower for the new set of condition functions from part (b) in comparison with the original set – part (a). This result is surprising because the new function set was evaluated to be able to realize a wide range of functions as described in Section II. Therefore, it is not possible to conclude that this function set would be generally more efficient than other sets. The lower success rate may be caused by the fact that the new function set in combination with the CMR approach represents a search space in which the target functions (regarding the multiplication task) are rather rare in comparison with the original set of condition functions. The optimal set of condition functions is thus evidently specific for a particular problem. Therefore, a more suitable function set may exist for solving the multiplication problem in cellular automata. However, to discover such optimal set still remains an open question.

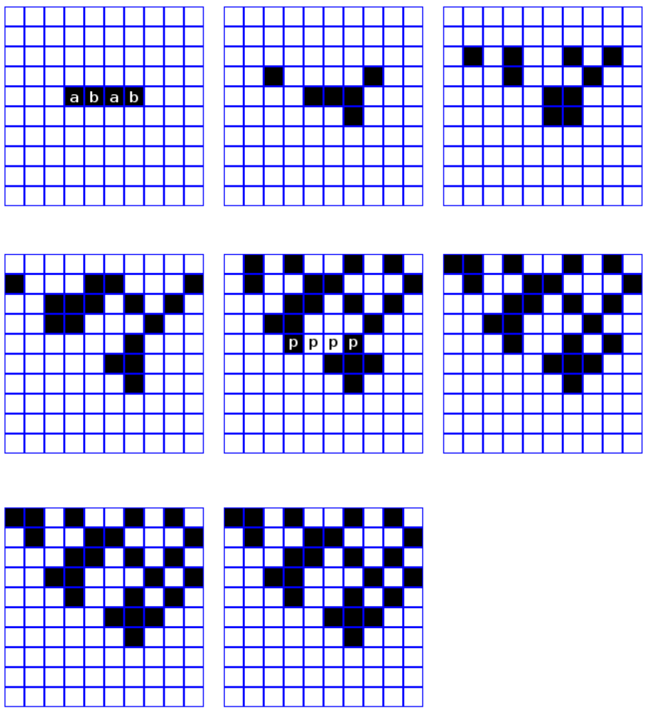


Fig. 4. Example of an evolved CA development performing the multiplication $3 \times 3 = 9$. White cell in the CA represents the state 0, black cell represents the state 1. The input operands as well as the resulting product are considered in direct binary representation using the cell states (the operands $a = b = (11)_{bin} = (3)_{dec}$, the product $p = (1001)_{bin} = (9)_{dec}$). Both the operand bits and product bits are interpreted from MSB to LSB as from left to right.

<=1	<=0	<=1	<=1	<=1	id	not	id	id	1
id	id	id	id	not	>=0	<=0	id	id	0
<=0	<=1	id	<=0	not	<=1	<=1	not	not	0
>=1	<=1	not	id	id	>=1	id	id	>=0	1
not	not	<=0	<=0	<=0	<=0	id	id	id	1
<=0	id	<=1	>=0	id	<=1	<=0	<=0	>=1	1

Fig. 5. Example of a CMR-based transition function discovered by GA using the function set from Table II. This transition function realizes multiplication in the CA from Figure 4. One row in this figure represents a CMR. The converted table-based representation consists of 31 conventional rules.

Nonetheless, both the proposed sets of condition functions allowed us to find some working solutions for different input and output cell configurations illustrated in Figure 3. As evident from Table III, the success rate is highly influenced by the cell setup. It can be seen that the highest success rates (greater than 60%) were achieved using the shared and alternating cell setups. The other setups (diagonal and separated) exhibit substantially lower success rates. The following reasons may be given for this result. If the input and output cells are close to each other, then the CA needs less steps to produce the result (i.e. the input values can be processed rather locally). On the other hand, if the inputs are distributed in a larger area or if the result is expected away from the inputs, then a more complex transition function is needed in order to process the inputs and transfer the resulting values to the output cells (the CA needs more steps to produce the result).

Figure 4 shows an example of evolved CA development performing the multiplication of two 2-bit operands. The appropriate CMR-based transition function discovered by GA is shown in Figure 5. The CA was initialized by the operand values within the input cells according to the shared cell setup shown in Figure 3d. In this case the CA needs 4 steps to produce the final product. As evident from Figure 4, the state of the output cells become stable during the subsequent CA development as required within the evolutionary process. In addition to the fact that the CA provides correct results for all the possible input operands, its development seems chaotic. However, an interesting emergent behavior can be observed when a larger CA is considered. For example, a final state of a 40x40-cell CA after the 30th development step is shown in Figure 6 with the input/output cells marked by p. The same task (i.e. to calculate the expression 3×3) was considered according to the setup from Figure 3d and the evolved transition function from Figure 5. Interestingly, the global CA behavior is similar to that produced by some typical rules of the elementary 1D cellular automata (for example, using the rules 22, 122 or 126 according to Wolfram code [3]). In our case, the pattern in Figure 6 was generated within a 2D CA by a successive "growth" of the cells upwards, developing a structure very similar to Sierpinski triangle. Although this structure can be obtained in a 1D CA using relatively simple rules from an initial seed, the transition functions obtained in this paper do not seem to be trivial. Similar emergent behavior can be observed within the development from a random initial state in a CA with enough area of 0-state cells available above the randomly initialized cells (see Figure 7). One of the open questions related to this issue is whether such emergent behavior could provide us with some advantages to perform (advanced) CA-based computation (e.g. using a specific operand encoding or interpretation of the CA development).

Another example of a successful multiplication in CA is illustrated in Figure 8 using the diagonal cell setup according to Figure 3c. This CA was discovered using the set of condition functions from Table I, the evolved transition function is shown in Figure 9. In this case the CA needs 10 steps to produce the result. After stabilizing the states of the output cells, the CA development exhibits an emergent pattern expanding from top to bottom and from left to right. In addition, a simple vertical line of state-1 cells grows upwards. We have determined that the number of those vertical lines depends on the number of 1's in the resulting product. This growing structure is very important in this particular CA because it contains the crucial states of the result. An example of the multiplication $3 \times 3 = 9$ is shown in Figure 10. As evident there are two 1s in the resulting product ($(9)_{dec} = (1001)_{bin}$) so that two state-1 lines grow upwards. The emerging pattern can be observed in the bottom part of Figure 10.

V. Conclusions

This paper presented a continuation of research regarding the evolutionary design of 2D uniform cellular automata

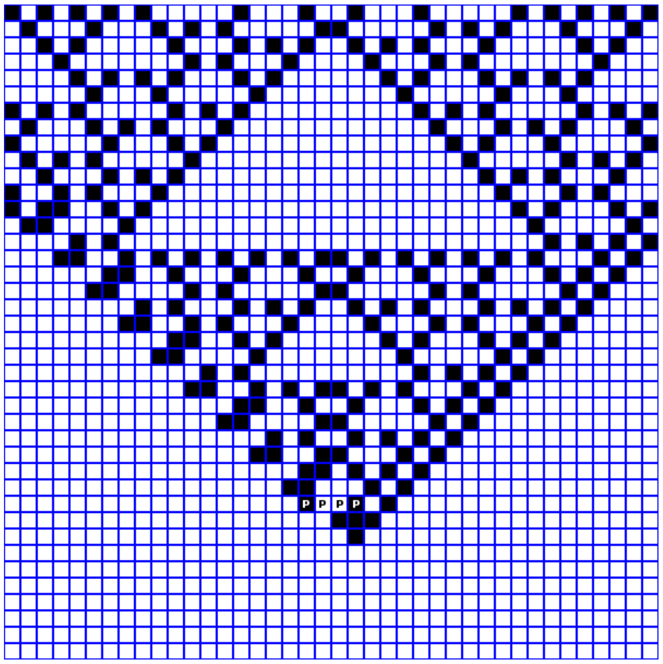


Fig. 6. The state of the 40x40-cell CA after the 30th development step performing the same multiplication task as shown in Figure 4 according to the transition function from Figure 5. The output cells containing the resulting product are denoted by p. These four cells were also initialized before the CA development by the values of the input operands. This is a final stable state of the CA that does not change anymore.

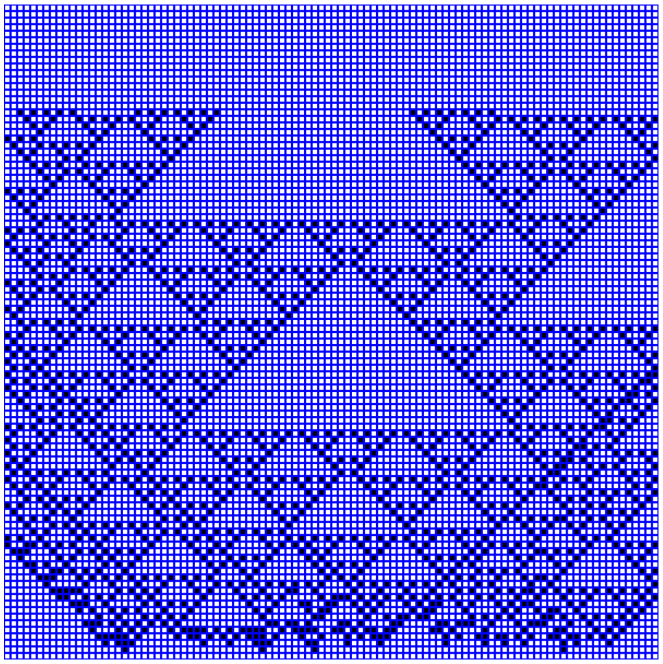


Fig. 7. An emergent pattern developed by 2D CA from some randomly initialized cells at the bottom of the cellular array. The CA develops according to the evolved transition function from Figure 5.

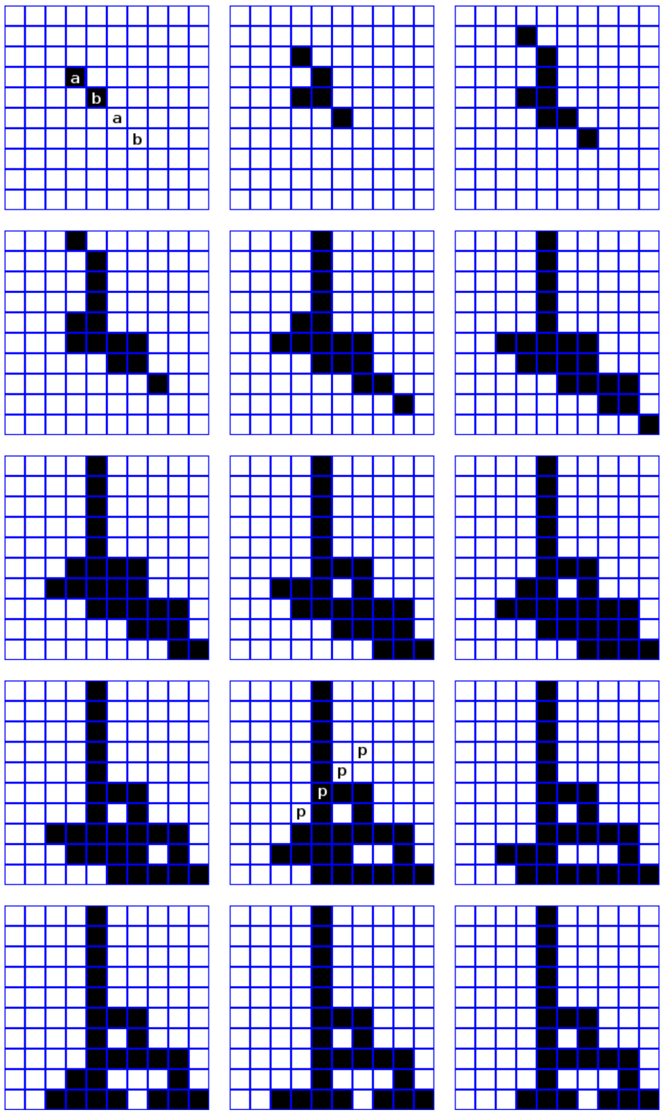


Fig. 8. Example of the development of CA performing the multiplication $2 \times 2 = 4$ using a diagonal input and output cell arrangement. White cell in the CA represents the state 0, black cell represents the state 1. The input operands are considered in the direct binary representation using the cell states ($a = b = (10)_{bin} = (2)_{dec}$, the product $p = (0100)_{bin} = (4)_{dec}$). The operand bits are interpreted at the main diagonal from MSB to LSB (from top-left to bottom-right), the product is interpreted at the antidiagonal from MSB to LSB (from bottom-left to top-right).

*	*	*	<=0	==0	*	>=1	*	<=0	0
==0	!=1	*	*	*	*	<=1	==1	!=1	1
!=1	==0	==0	<=1	*	*	>=0	<=1	*	0
==1	>=1	!=0	==1	*	<=1	>=1	*	>=1	0
!=0	<=0	*	*	>=0	<=0	<=0	==0	<=1	1
<=1	!=1	<=0	*	<=0	>=0	!=1	>=0	==1	1
!=1	<=0	==1	*	==1	!=0	==1	*	*	0
>=0	>=0	*	!=1	*	!=0	*	*	==0	1

Fig. 9. Example of a CMR-based transition function discovered by GA using the function set from Table I. This transition function realizes multiplication in the CA from Figure 8. One row in this figure represents a CMR. The converted table-based representation consists of 73 conventional rules.

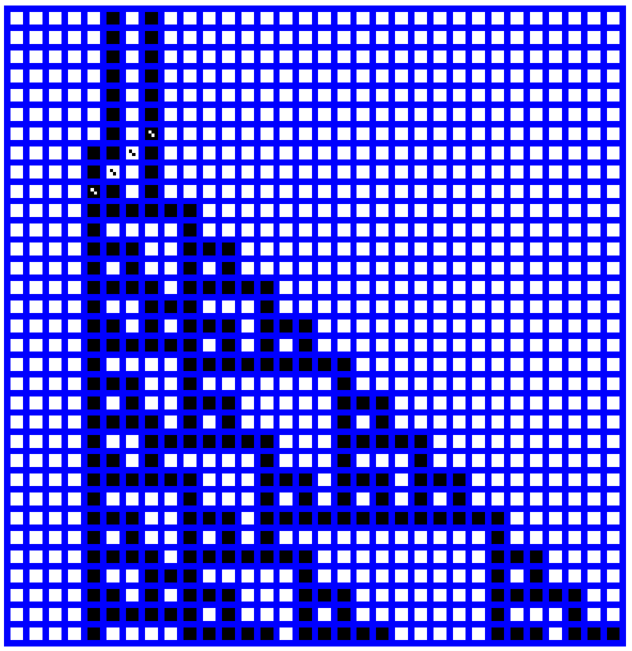


Fig. 10. Emergent behavior of the CA controlled by the transition function from Figure 9. The result of the product $3 \times 3 = 9$ is shown within the cells marked by two dots in the upper-left part of the CA. The diagonal cell setup was considered according to Figure 3c.

using conditionally matching rules. We focused on a case study whose objective was to design CA that are able to perform multiplication of two 2-bit operands. Several sets of experiments were presented considering various setups of both the evolutionary system (using the genetic algorithm) and cellular automata. In particular, two different sets of condition functions for realizing the transition rules of the CA were investigated in combination with various input and output cell arrangements in the CA. The experiments showed that the success rate of discovering a working solution is highly dependent on that cell arrangement. Moreover it also depends on the set of functions considered for the conditionally matching rules. Surprisingly, the experiments exhibit lower success rate in case of the condition functions that were identified in a separate experiment as potentially promising to solve a wide range of tasks. However, all sets of experiments provided some working solutions with interesting features.

The first is that the evolved transition functions are not limited to a CA of a specific size (i.e. the considered multiplication task may perfectly be solved in a larger CA even with more areas with the input and output cell structures). The second interesting phenomenon is the emergent global behavior (developed patterns) of the resulting CA in which a well-organized structures can be observed in addition to the primary computational task. A detailed analysis of some evolved results showed that such well-organized patterns of various complexity is a common feature of CA exhibiting the required computation. An interesting fact of these patterns is that they are very similar to some typical structures generated by one-dimensional cellular automata.

One of the questions that may arise from this investigation may be whether the process of multiplication could be simplified or optimized (e.g. to minimize the number of rules of the converted conventional transition function). Since massive parallel and homogeneous platforms may be very important in the future (especially in nanotechnology, molecular computing systems and similar areas), the principles of elementary approaches to studying the design, optimization and functioning of such systems are definitely worth the subsequent research.

The observations from the obtained results propose some issues for the future research in this area. For example, the selection of optimal set of condition functions in order to effectively design CA for solving a specific task still represents a challenging task. The in-depth analysis of the evolved functions and the possibilities of their applications in a wider context of the cellular platforms will also be investigated. Our ongoing experiments are devoted to the research of conditionally matching rules for efficient design of non-binary cellular automata.

Acknowledgements

This work was supported by the Czech Science Foundation project 14-04197S, BUT project FIT-S-14-2297 and the IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070, funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

References

[1] J. von Neumann, *The Theory of Self-Reproducing Automata*. A. W. Burks (ed.), University of Illinois Press, 1966.
[2] M. Bidlo and Z. Vasicek, "Evolution of cellular automata with conditionally matching rules," in *2013 IEEE Congress on Evolutionary Computation (CEC 2013)*. IEEE Computer Society, 2013, pp. 1178–1185.
[3] S. Wolfram, *A New Kind of Science*. Champaign IL: Wolfram Media, 2002.
[4] M. Sipper, *Evolution of Parallel Cellular Machines – The Cellular Programming Approach, Lecture Notes in Computer Science, volume 1194*. Berlin: Springer-Verlag, 1997.
[5] G. Tufte and P. C. Haddow, "Towards development on a silicon-based cellular computing machine," *Natural Computing*, vol. 4, no. 4, pp. 387–416, 2005.
[6] S. Kumar and P. J. Bentley (eds.), *On Growth, Form and Computers*. Elsevier Academic Press, 2003.
[7] E. F. Codd, *Cellular Automata*. Academic Press, New York, 1968.
[8] K. Lindgren and M. G. Nordahl, "Universal computation in simple one-dimensional cellular automata," *Complex Systems*, vol. 4, no. 3, pp. 299–318, 1990.
[9] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays, Vol. 2*. A. K. Peters/CRC Press, 1982.
[10] C. G. Langton, "Studying artificial life with cellular automata," *Physica D: Nonlinear Phenomena*, vol. 22, no. 1–3, pp. 120–149, 1986.
[11] A. Gajardo, A. Moreira, and E. Goles, "Complexity of langton's ant," *Discrete Applied Mathematics*, vol. 117, no. 1–3, pp. 41–50, 2002.
[12] C. G. Langton, "Self-reproduction in cellular automata," *Physica D: Nonlinear Phenomena*, vol. 10, no. 1–2, pp. 135–144, 1984.
[13] G. Tempesti, "A new self-reproducing cellular automaton capable of construction and computation," in *Advances in Artificial Life, Proc. 3rd European Conference on Artificial Life*, ser. Lecture Notes in Artificial Intelligence, vol. 929. Springer Verlag, 1995, pp. 555–563.