Evolution of Cellular Automata Using Instruction-Based Approach

Michal Bidlo
Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Božetěchova 2, 61266 Brno
Czech republic
Email: bidlom@fit.vutbr.cz

Zdenek Vasicek
Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Božetěchova 2, 61266 Brno
Czech republic
Email: vasicek@fit.vutbr.cz

***Abstract*—This paper introduces a method of encoding cellular automata local transition function using an instruction-based approach and their design by means of genetic algorithms. The proposed method represents an indirect mapping between the input combinations of states in the cellular neighborhood and the next states of the cells during the development steps. In this case the local transition function is described by a program (algorithm) whose execution calculates the next cell states. The objective of the program-based representation is to reduce the length of the chromosome in case of the evolutionary design of cellular automata. It will be shown that the instruction-based development allows us to design complex cellular automata with higher success rate than the conventional table-based method especially for complex cellular automata with more than two cell states. The case studies include the replication problem and the problem of development of a given pattern from an initial seed.**

***Index Terms*—Cellular automaton, development, replication, evolutionary design.**

I. Introduction

In the recent years cellular automata (CA) have been successfully applied in many scientific areas. The development of a cellular automaton usually represents a complex process during which a non-trivial global behavior based only on local cell interactions using simple rules may emerge [1]. However, the design of a transition function according to which the CA should develop to solve a given problem is a challenging task. The problem is that the number of possible solutions grows exponentially with the increasing number of cell states and the size of the cellular neighborhood. Moreover, the process of creating the transition function is less intuitive than the traditional algorithm design because of local cell interactions and parallel matter of the CA development. Therefore, non-traditional approaches have often been applied, including evolutionary algorithms.

The goal of this paper is to introduce an instruction-based approach for the development of cellular automata. The main idea is to represent the transition function by a program (a sequence of instructions performing simple elementary operations) rather than by a table specifying a new state of a cell for all the possible combinations of states in the cellular neighborhood. It will be shown that by using the instruction-based approach the transition function for a given problem may be designed in substantially shorter time and with higher success rate in comparison with the conventional (table-based) approach. The experiments performed to demonstrate the ability of the proposed approach consider the replication problem and the development of a specified pattern in the cellular automaton. The simple genetic algorithm will be utilized to design the cellular automata.

The paper is organized as follows. The rest of this section briefly introduces the basic principles of cellular automata and summarizes the related work. In Section II the concept of instruction-based development for cellular automata is described. The setup of the evolutionary system utilized for the experiments is stated in Section III. Overview of the experimental results and discussion is proposed in Section IV. Finally, Section V provides concluding remarks and possible direction of future research.

A. Cellular Automata

Cellular automata, originally invented by Ulam and von Neumann in 1966 [2], represent a mathematical model intended to study the behavior of complex systems, especially the questions of whether computers can self-replicate. Cellular automata may also be considered as a biologically inspired technique to model and simulate the cellular development. A two-dimensional (2D) cellular automaton consists of a regular grid of cells, each of which can occur in one state from a finite set of states. The states are updated synchronously in parallel according to a local transition function. The synchronous update of all the cells of the CA is called a developmental step. The next state of a cell depends on the combination of states in the cellular neighborhood. In this paper the cellular neighborhood will be considered as a 5-tuple comprising the investigated cell and its immediate neighbor in the north, south, east and west direction. The standard form of the transition function defines next state of a given cell for every possible combination of states in its neighborhood. Let us denote $s_N s_S s_E s_W s_C \rightarrow s_{Cnew}$ a rule of the transition function, where s_N, s_S, s_E, s_W and s_C represents the actual state of the north, south, east, west and the central cell in the cellular neighborhood respectively and s_{Cnew} denotes the next

state of the investigated (central) cell. This concept is referred to as von Neumann's cellular neighborhood consisting of 5 cells. Boundary conditions have been considered for a finite size of the cellular grid. Typically zero boundary conditions have been applied which means that the non-existing neighbors of the cells at the grid boundary are considered as cells in state 0. Another case may involve cyclic boundary conditions, i.e. the opposite cells at the grid boundary are considered to be neighbors and then the 2D CA can be viewed as a toroid. In case of uniform cellular automata the transition function is identical for all the cells. In general, non-uniform CA may have each cell driven by different transition function.

In this paper 2D uniform cellular automata with von Neumanns neighborhood and cyclic boundary conditions will be considered.

B. Related Work

Cellular automata have been applied to solve many complex problems in different areas. A detailed survey of the principles and analysis of various types of cellular automata and their applications is summarized in [1]. Sipper [3] investigated the computational properties of cellular automata and proposed an original evolution-based method called cellular programming for the design of non-uniform cellular automata. He demonstrated the success of this approach in solving some typical problems related to the cellular automata, e.g. synchronization task, ordering task or the random number generation. In the recent years, scientists have been interested in the design of cellular automata for solving different tasks using the evolutionary algorithms.

Several works dealt with the replication problem in the past as well as in the recent years. Many works have dealt with the design and development of cellular automata or more general cell-based systems (e.g. Random Boolean Networks [4]). For example, Miller investigated the problem of evolving a developmental program inside a cell to create multicellular organisms of arbitrary sizes and characteristics. He presented a system in which the organism organizes itself into a well defined patterns of differentiated cell types (e.g. the French flag) [5]. Kowaliw et al. proposed a simplified model of biological embryogenesis instantiating a subset of 2D cellular automata and a methodology for "growing" the cells into agents utilizing only local interactions. His approach was called Bluenome Developmental Model [6]. Tufte and Haddow utilized a FPGA-based platform of Sblocks [7] for the online evolution of digital circuits. The system actually implements a cellular automaton whose development determines the functions and interconnection of the Sblock cells in order to realize a specified behavior [8]. The rules for the development of the cellular automaton has been designed by evolutionary algorithm. Considering the popular replication problem, probably the most known approach represents the Langton's self-replicating loops [9] that utilize special instructions encoded in the cell states to determine the development steps of the cellular automaton. In particular, the loop starts its replication by creating a "construction arm" by means of which the new copy of itself emerges. The instruction specified by the combinations of states in this arm determines the next step of the replication process (including turns, loops closing and starting the next replication process). Pan and Regia also studied the replication in cellular automata [10]. However, they adopted a uniform tree-based approach based on Genetic Programming for representing both arbitrary cellular automata structures and the rules that control the cell's transitions. As the authors state "There is no identifiable instruction sequence or construction arm, the replicating structures generally translate and rotate as they reproduce, and they divide via a fissionlike process that involves highly parallel operations." [10]. We found their approach very inspirative because it actually introduces new way of determining the states during the CA development. However, we also felt that the method utilized to calculate the transition function might be simplified substantially by introducing elementary operations and suitable encoding with respect to the form of the cellular neighborhood. As we demonstrate, our approach is applicable on different problems in two-dimensional cellular automata.

II. Instruction-Based Development for Cellular Automata

The instruction-based development (IBD) was originally introduced in [11] as an advanced generative genotype–phenotype mapping in the evolutionary design. The main goal was to provide an evolutionary system for the automatic development of generic solutions for different problems. The instruction-based approach demonstrated its ability to reduce the search space allowing to develop (arbitrarily) large structures (instances) of digital circuits.

However, the concept of instructions also may be utilized for effective representation of functions (similarly to Genetic Programming for the evolution of computer programs [12]). Cellular automata belong to the systems in which an efficient calculation of the local transition function (determining the process of their development) is essential to solve a given problem. 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. In case of increasing the number of cell states the number of such combinations grows exponentially and thus the representation and design of the transition function becomes difficult. It may me possible to specify implicit rules of the transition function (e.g. for some combinations of states the new state of the cell does not change) but the problem is how to determine the set of implicit rules for a given task. Therefore, we will represent the transition function by a program whose instructions perform elementary or more complex operations over the cell states of the cellular neighborhood and the next state is chosen deterministically from this modified neigborhood. Whilst in [11] the instructions were intended to manipulate the circuit building blocks (i.e. to perform a construction process), in this paper another instruction set has to be chosen. In particular, the instructions will be devoted to the calculation over cell states and other operations related to the cellular neighbor-

hood. The main idea is to demonstrate that the instruction-based approach combined with evolutionary algorithms may be widely applicable. In this paper the case studies include some problems of cellular automata development, specifically the replication problem and the development of a given pattern from an initial seed. The objective is to show that if a suitable set of instructions is utilized for the evolution of a program-based transition function of a cellular automaton, then a given behavior of the CA can be achieved with higher success rate in comparison with the conventional table-based representation.

A. Operations on the cellular neighborhood

The goal of the IBD approach to cellular automata evolution is to provide a technique for efficient updating the cell states during the CA development with respect to the states of the neighboring cells. The operations of the instructions have been chosen with respect to the form of the cellular neighborhood. The execution of the program allows to modify the states in the cellular neighborhood and subsequently to determine the next state of the investigated cell. The following development algorithm will be considered for each cell of the CA:

1) Copy the cell states of the cellular neighborhood into a temporary data structure whose form corresponds to the cellular neighborhood.
2) Execute the program representing the transition function whose instructions will modify the states in the temporary data structure.
3) Return the state of the central cell in the temporary data structure as the next state – the result of the transition function.

The set of instructions that may be utilized in the program calculating the transition function is summarized in Table I. As evident the instructions include operations that can modify one or more cells in the neighborhood copy and the empty operation allowing to alter the efficient length of the program during the evolutionary process. Since the instructions operate over the copy of the cellular neighborhood in a temporary data structure, the process of calculation of the next state of a cell does not influence the states of other cells during a development step and therefore the next states of all the cells can be determined in parallel which is a characteristic feature of cellular automata. The instructions were chosen with respect to general operations that are possible to perform over cell states (i.e. logic and arithmetic operations over the state values, transfer a cell state to a different cell in the neighborhood, swapping the states of two neighbors etc.). However, no advanced optimization of the instruction set has been performed in this stage of research because the selection of proper instructions for a given CA behavior represents a difficult task and in many cases is a subject of experimental work.

B. Properties of the Instruction-Based Transition Function

If an evolutionary algorithm is applied to design a CA, the instruction-based approach is able to shorten the chromosome substantially and therefore to reduce the search space. In fact, the design of a CA consists of the evolution of its local transition function.

TABLE I
THE SET OF INSTRUCTIONS UTILIZED FOR THE DEVELOPMENT OF CELLULAR AUTOMATA. $N[i_1], N[i_2]$ DENOTE THE CELL STATES FROM THE NEIGHBORHOOD POSITIONS DETERMINED BY THE INSTRUCTION ARGUMENTS i_1, i_2, S REPRESENTS THE NUMBER OF CELL STATES AND N, S, E, W AND C SPECIFIES THE CELL STATE IN THE NORTH, SOUTH, EAST, WEST AND CENTRAL POSITION IN THE NEIGHBORHOOD RESPECTIVELY.

Instruction	Operation	Description
AND	$N[i_1] = N[i_1] \wedge N[i_2]$	logic AND
OR	$N[i_1] = N[i_1] \vee N[i_2]$	logic OR
XOR	$N[i_1] = N[i_1] \oplus N[i_2]$	logic XOR
NOT	$N[i_1] = notN[i_1]$	bitwise NOT
INV	$N[i_1] = S - N[i_1]$	inverse state
MIN	$N[i_1] = min(N[i_1], N[i_2])$	minimum
MAX	$N[i_1] = max(N[i_1], N[i_2])$	maximum
SET	$N[i_1] = N[i_2]$	replace
INC	$N[i_1] = N[i_1] + 1$	increment
DEC	$N[i_1] = N[i_1] - 1$	decrement
SWP	$N[i_1] \leftrightarrow N[i_2]$	swap
ROR	$WCE \rightarrow EWC$	rotate right
ROL	$WCE \rightarrow CEW$	rotate left
ROU	$UCS \rightarrow CSU$	rotate up
ROD	$UCS \rightarrow SUC$	rotate down
NOP		no operation

For example, if a transition function ought to be evolved for a CA working with 4 cell states (that is used in some of the experiments presented in Section IV), then the fully defined table-based transition function consists of $4^5 = 1024$ integers (it is the length of a chromosome representing the complete table of the transition function). Therefore, there are in total $4^{1024} = 3.2317 \times 10^{616}$ different transition functions for this CA which represents the search space of the evolutionary algorithm. Consider that the IBD approach is utilized and the goal is to evolve a program consisting of 10 instructions. Moreover, assume that a single instruction consists of 3 integers (operation code and two arguments), there are 16 different instructions (i.e. 16 different operation codes) and each of the arguments can posses one of 5 different values. Then the length of a chromosome is $10 \times 3 = 30$ integers and the size of the search space consists of $(16 \times 5 \times 5)^{10} = 1.048576 \times 10^{26}$ different programs which is substantially less in comparison with the table-based representation.

As stated in the previous section, the program is executed over a copy of the cellular neighborhood. Therefore, the next states of all the cells can be calculated independently (in parallel) as usual in common (synchronous) cellular automata. Another important aspect of the IBD approach is that the process of calculating the next state for a given cell is deterministic (there is a specific combination of states in the neighborhood copy which the program operates on, each instruction of the program performs a deterministic operation (function) modifying the states in the neighborhood and the resulting value — next state — is always considered in a specific cell of the neighborhood after executing the program). Considering this feature, the instruction-based transition function can be deterministically transformed to a corresponding

table-based transition function without changing the nature of cellular automata.

III. Evolutionary System Setup

The simple genetic algorithm (GA) was utilized for the evolutionary design of the cellular automaton that exhibits the specified behavior. For the comparison purposes we consider the evolution of common table-based local transition function as well as the program-based transition function as described in the previous section. The table-based approach considers the evolution of a complete transition function (i.e. to determine a next state for all the possible combinations of states in the cellular neighborhood). In case of the IBD approach a program to be evolved consists of 10 instructions. This value was determined experimentally in order to provide a sufficient resources to calculate the next states. Of course, some of the resulting solutions use NOP instructions so the effective length of the program can be reduced. However, if shorter programs ought be evolved, then the number of correct solutions in the search space may be reduced and the success rate decreases.

In all the experiments, the population consists of 16 chromosomes which are initialized randomly (with respect to the correct range of each gene) at the beginning of evolution. The chromosomes are selected by means of the tournament operator with the base 4. The experiments showed that the crossover operator is not suitable for this problem, thus only the mutation operator is applied as follows. Two integers of the chromosome are chosen randomly and their values are mutated by generating new random values in the appropriate range.

Each candidate CA is evaluated during 30 development steps according to the transition function encoded in the chromosome. The following subsections describe the specific features of the evolutionary system with respect to the two different approaches.

The initial state of the CA, the way of calculating the fitness function and the number of generations of the evolution depends on the problem to be solved and therefore their description will be covered in Section IV.

The way of encoding the transition function in the genome for the table-based and program-based representation and its properties is described in the following subsections.

A. Table-Based Transition Function

In case of the table-based transition function the chromosome encodes the next states of a cell for all the possible combinations of states in the cellular neighborhood. The index of a given next state in the chromosome is specified implicitly by means of the value expressed by the number representing the combination of states in the cellular neighborhood. The base of this number equals the number of possible cell states. Therefore, if we consider the general form of the rule $s_N\ s_S\ s_E\ s_W\ s_C \rightarrow s_{Cnew}$, only the part on the right of the arrow are encoded in the chromosome. For example, if a cellular automaton ought to be evolved working with 2 different cell states and von Neumann's neighborhood consisting of 5 cells, there are 2^5 rules of the local transition function. Consider the rule $1\ 0\ 0\ 0\ 1 \rightarrow 0$. Since the combination of states $1\ 0\ 0\ 0\ 1$ corresponds to the binary representation of value 17, the output value (0) will be placed in the chromosome at the position 17.

B. Program-Based Transition Function

The program-based representation of the transition function is encoded in the chromosome as a finite sequence of instructions from Table I. Each instruction is encoded as three integers (operation code and two arguments) whose value ranges depend on the number of instructions and the meaning of their arguments. The main advantage of this approach is that the length of the genome is independent on the number of cell states and the size of the cellular neighborhood. Therefore the search space can be reduced substantially.

IV. Experimental Results and Discussion

The abilities of the proposed instruction-based development approach introduced in the previous sections will be demonstrated on two problems: (1) the replication problem and (2) the problem of development of a given pattern from a seed. The experimentsl results and discussion are given in this section.

A. Replication Problem

The goal of the replication problem is to obtain a copy of a given structure in a finite number of development steps. The structure is represented by the initial state of the CA. The genetic algorithm is applied to design a transition function by means of which the CA develops so that there is a given number of copies of the initial structure after a finite number of development steps. The set of experiments performed in this section considers searching for the transition function (in the form of table and program) for the replication of structures of different complexity and size. As noted in Section I-B there are several approaches to the replication problem. Probably the simplest technique able to replicate an arbitrary structure is based on additive cellular automata rules [1]. This problem can be viewed as a basis for investigating the abilities of the proposed method (having a known solution, we may search for the same or similar transition functions using the conventional and proposed approach).

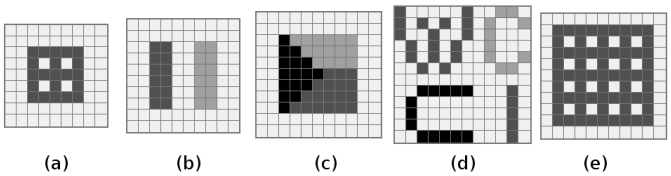

Fig. 1. Patterns considered in the experiments.

Five sets of experiments were performed, each of which contained 100 independent runs of the GA. The first set considered the replication of a simple grid structure (Figure 1a), the second was devoted to the replication of French flag pattern (Figure 1b), the third set is the replication of Czech flag

```
best_fitness = 0  # fitness out of all development steps
const REPLICS = the num. of required copies of the given pattern

initialize the CA by the pattern to be replicated
FOR int step = 1 TO DEVEL_STEPS DO
{
   fitness = 0  # fitness in one development step
   replicas_cnt = 0  # num. of replicas found in a devel. step
   ca_step(cal, genome->prog);

   FOR row = 0 TO CA_HEIGHT - PATTERN_HEIGHT DO
   {
      FOR col = 0 TO CA_WIDTH - PATTERN_WIDTH DO
      {
         partial_fitness = 0  # fitness in specific part of CA
         FOR pr = 0 TO PATTERN_HEIGHT - 1 DO
            FOR pc = 0 TO PATTERN_WIDTH - 1 DO
               IF ca[row+pr][col+pc] == pattern[pr][pc] THEN
                  partial_fitness = partial_fitness + 1
         save the partial_fitness value
         IF found perfect pattern at position (row, col) THEN
            replicas_cnt = replicas_cnt + 1
      }
   }
   fit = sum of the REPLICS best saved partial fits
   # add a bonus if the solution produces more replicas
   fit = fit + replicas_cnt * PATTERN_HEIGHT * PATTERN_WIDTH

   # save the best fitness out of all development steps
   IF fitness > best_fitness THEN
      best_fitness = fitness
}

RETURN best_fitness
```

Fig. 2. Calculating the fitness function for the replication problem. The pattern dimensions $PATTERN\_WIDTH$ and $PATTERN\_HEIGHT$ also include a border consisting of a single line of inactive cells (cells in state 0) because we require the replicated structures to be separated each other.

TABLE II
STATISTICAL RESULTS FOR THE REPLICATION PROBLEM CONSIDERING THE INSTRUCTION-BASED AND TABLE-BASED DEVELOPMENT. THE GRID STRUCTURES (FIG. 1A) WERE DEVELOPED IN CA WITH 2 CELL STATES, THE OTHER PATTERNS CONSIDERED 4 CELL STATES. IF NOT EXPLICITLY SPECIFIED, 3 REPLICAS WERE REQUIRED.

Instruction-based development						
		Number of generations				
Pattern	**Succ** [%]	**avg.**	**std. dev.**	**min.**	**median**	**max.**
Fig. 1a	100.0	23	19.4	1	18	80
Fig. 1b	100.0	22	16.4	1	19	80
Fig. 1c	100.0	22	21.4	1	18	112
Fig. 1d	100.0	23	18.1	1	18	81
Fig. 1d (4 repl.)	100.0	55	43.6	2	47	256
Table-based development						
Fig. 1a	9.0	5634	2490.2	1500	5250	9052

(Figure 1c), the fourth and fifth replicate WCCI abbreviation (Figure 1d), where at least 3 and 4 copies are required respectively. The evolution was executed independently for the design of the conventional table-based transition function and the program-based representation. The algorithm calculating the fitness function is shown in Figure 2 and its principle can be described as follows. After each development step, every part of the CA is explored by comparing the states of the given pattern with the appropriate cell states at the corresponding positions in the CA. If a state match is detected, then a partial fitness value associated with the specific part of the CA is increased by one. After exploring the part of the CA the resulting partial fitness is saved into a temporary array. If the partial fitness equals the number of cells the replicating pattern is composed of, then the value of a replicas counter variable is increased by one. After exploring all the parts of the CA, the replicas counter contains the number of perfectly matched patterns (i.e. the number of replicas that emerged after a given development step). The fitness value of the given development step is calculated as the sum of the $REPLICS$ best partial fitness values, where $REPLICS$ represents the number of required copies of the (initial) pattern to be replicated. If the replicas counter detected at least one replicated pattern, then the fitness value is increased appropriately to prefer solutions that are able to create perfect replicas. As a final fitness value of the CA (i.e. the fitness of the candidate transition function) is considered the highest fitness from all the development steps during which the CA was evaluated.

The results of the replication experiments are summarized in Table II. For each pattern the success rate and the average number of generations needed to find a perfect solution were measured. The proposed program-based transition function overcomes the conventional approach in all presented cases, especially for more complex patterns. In case of the grid structure replication a perfect program was evolved in all runs, whilst the table-based approach succeeded only in 9% of evolutionary runs. It is important to note that the table-based approach did not provide any solution to the remaining patterns considered in the experiments. We assume that this result is caused by the cardinality of the search space that is substantially higher for the table-based representation and the evolution is not able to explore it effectively. Another aspect of this issue is probably based on the operations needed to express the local transition function. In case of the table-based representation, the transition function actually needs to be created at a low level (i.e. for every combination of states in the cellular neighborhood a new state has to be specified). However, if the instruction-based approach is considered, the new state is calculated using higher-level operations (like in a common programming language), the corresponding program can be shorter in comparison with the complete table which leads to a reduction of the search space and the evolution is able to explore it more effectively. We determined that several different programs were evolved that produces at least 3 perfect replicas of the given pattern. Although we required 3 replicas, the evolution found in some case a solution providing 4 replicas of the structure.

An example of evolved solution is shown in Figure 3. In addition, the table-based transition function produced two solutions for 3 replicas that exhibit a couple of extra active cells between the replicated patterns (Figure 4). This behavior was not observed in the program-based approach (only pure replicated structures were generated). It is probably caused by the fact that the evolution of the table-based representation directly allows to alter each single output state of the transition function whilst the program-based approach actually represents an indirect mapping between the input combinations of states in the cellular neighborhood and the output states. This feature may be considered as both advantage and disadvantage of the program-based approach. The benefit lies in the fact that the program-based solutions produce perfect outputs without

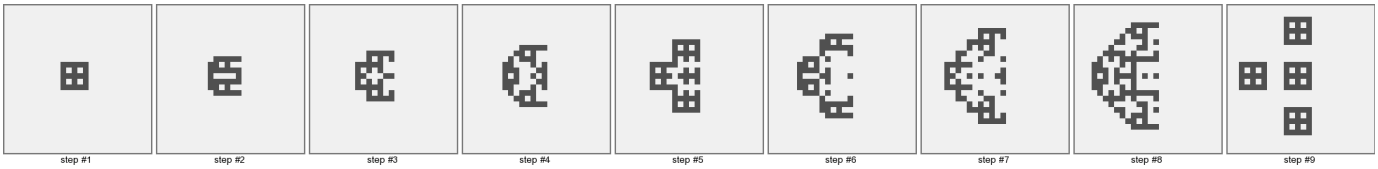

Fig. 3. Development of evolved cellular automaton for the replication of a grid structure.

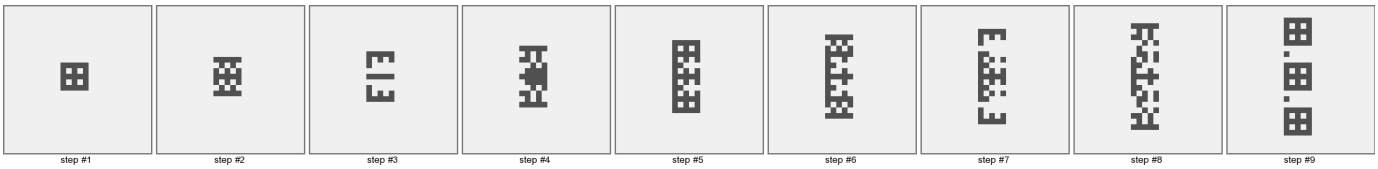

Fig. 4. Example of replication of a simple grid structures with additional active cells.

undesirable active cells. On the other hand the drawback is that more complex transition functions require (as expected) more instructions in the program. Nevertheless the evolution is able to tackle that very efficiently because the proposed approach solved all the considered problems with substantially higher success rate and lower computational effort in comparison with the table-based transition function.

Another examples illustrating the replication of more complex irregular patterns (the Czech flag and a WCCI pattern) are illustrated in Figure 5 and 6. Both of these automata operate with 4 cell states.

The evolved transition functions exhibit the features of additive rules described in [1]. Several different variants were obtained differing in the number and direction of replicas with respect to the position of the initial structure. In addition to the pattern used during the evolution, the resulting programs are in many cases able to replicate different structures which confirms the properties of the replicators mentioned in [1].

B. Pattern Development Problem

Another issue that was investigated in our experiments is the problem of the development of a given pattern in a cellular automaton from a seed. It means that the initial state of the CA is represented only by the central cell in non-zero state, all the other cells possess the state 0. During the evolutionary process the CA is examined if it matches with the specified pattern after each development step. In these experiments the dimensions of the cellular automaton correspond to the dimensions of the pattern that ought to be developed. The goal is to design a transition function (again, in the form of table and program) according to which the CA develops from the seed into the given pattern.

Four sets of experiments were performed. The first pair of experiments considered the development of a grid structure consisting of 5x5 cells (Figure 1a)) and 9x9 cells (Figure 1e). In the second pair of experiments French flag ought to be developed (Figure 1b) with the dimensions 6x6 and 9x9 cells. The candidate solutions are evaluated as follows. A partial fitness value is calculated after each development step as the number of cells of the CA whose state equals the state of the corresponding cell of the target pattern. The fitness function of a candidate transition function is evaluated as the maximum of the partial fitness values from all the development steps.

TABLE III
STATISTICAL RESULTS FOR THE PATTERN DEVELOPMENT PROBLEM CONSIDERING THE INSTRUCTION-BASED AND TABLE-BASED APPROACH. THE GRID STRUCTURES (FIG. 1E) WERE DEVELOPED IN CA WITH 2 CELL STATES, THE OTHER PATTERNS CONSIDERED 4 CELL STATES.

Instruction-based development						
Pattern	**Succ. [%]**	**Number of generations**				
		avg.	**std. dev.**	**min.**	**median**	**max.**
Fig. 1a	100.0	14358	16711.5	143	7543	86445
Fig. 1e	60.0	32504	23387.7	2888	24828	89228
Fig. 1b	79.0	37925	27117.1	1211	31991	97717
French9x9	23.0	62095	21979.8	18784	62143	90233
Table-based development						
Fig. 1a	100.0	402	757.3	19	118	4075
Fig. 1e	76.0	24331	26200.6	118	16353	96980
Fig. 1b	54.0	28896	26264.1	475	22028	92948
French9x9	1.0	30614	0.0	30614	30614	30614

Table III summarizes the statistical results from the experiments mentioned in the previous paragraph. The evolution succeeded in all cases and provided solutions that perfectly fulfil the objectives specified in the fitness function. There are some interesting facts that were observed in both representations of the transition function. The first is that the instruction-based approach exhibits higher success rate in most sets of experiments. The only case in which the table-based representation is more successful is the development of a 9x9 grid structure (the program-based approach succeeded in 60% whilst the conventional method in 76%). This issue can be explained as follows. The problem considers a binary CA whose 5-neighborhood implies 2^{32} possible transition functions specified by the table (the chromosome consists of 32 bits). However, the search space of the program-based approach is in this case substantially bigger. For example, if 10 instructions in the programs are considered, each consisting of 3 integers, there are 30 integers in the chromosome, each of which can possess at least 5 different values so the search

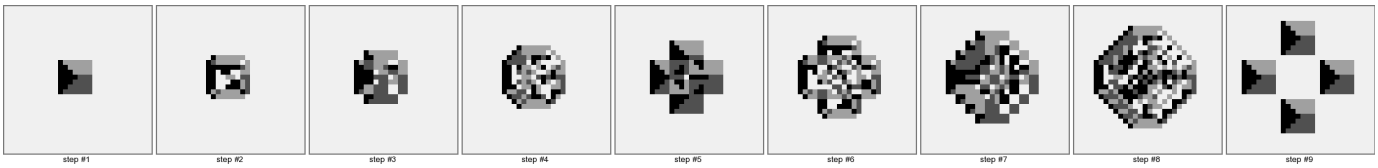

Fig. 5. Development of evolved cellular automaton for the replication of the Czech flag.

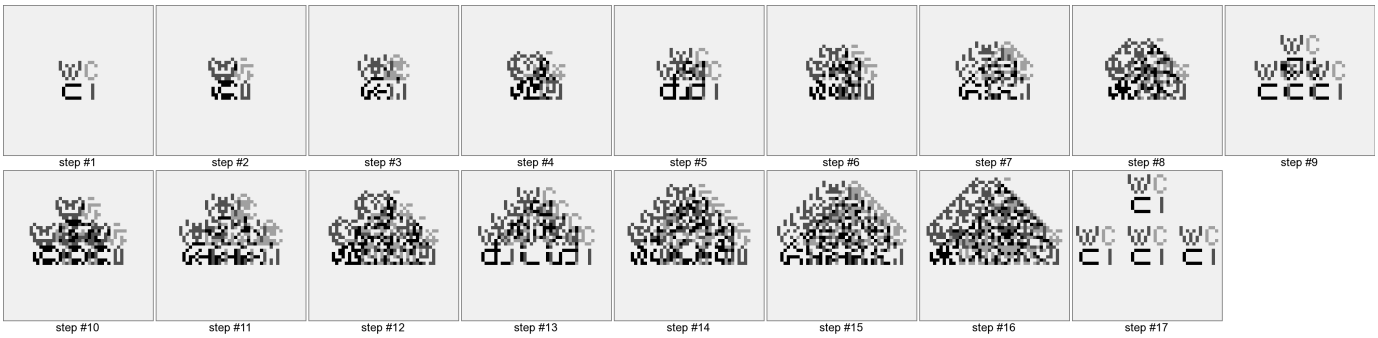

Fig. 6. Development of evolved cellular automaton for the replication of WCCI structure.

space contains at least 5^{30} candidate solutions. Therefore it is harder to find a working solution for the 9x9-cell structure in so big search space. The second interesting issue is that although the program-base approach mostly exhibits higher success rate, the computational effort (expressed by the number of generations needed to evolve a working solution) is higher than in case of the conventional approach. This fact was observed in all the experiments performed in the pattern development problem. We assume that this feature is caused by more complex (indirect) mapping between a program and the corresponding output states of the transition function of the cellular automata.

Figure 7 shows an example of evolved solution for the development of French flag in a cellular automaton. In this case we obtained several solutions that differ in the behavior of the developed structure if the CA continues to develop. In most cases the French flag pattern represents an intermediate state of the CA that is totally destroyed if the development continues. The second group of solution is able to periodically recreate the given pattern and the last case includes several solutions that produce the French flag that is stable during the subsequent development of the CA. These classes of solutions are expectable. Since the CA possesses finite dimensions and the number of cell states is also finite, it can not exhibit infinite development through infinite different states. Therefore, if the CA does not exhibit a stable pattern after a finite number of development steps, then it generates a finite number of different patterns in a loop (e.g. see Figure 7)). The corresponding program that was found by evolution is shown in Table IV. It is very difficult to identify the principle of this program (similarly as to identify the individual rules of a transition function) because the CA behavior is an emergent property of interaction of all the cells. It can be observed that all the (temporary) neighborhood cells are affected by the program so that the development of French flag is probably not a trivial task. Note that the exact French flag pattern was reached only in CA whose dimensions correspond to the pattern size. In larger CA, although, it is possible to develop the pattern in a subpart of the CA but some of the other cells are affected too that surrounds the target pattern immediately (confirmed by the experiments).

TABLE IV
EVOLVED 6X6 CELLULAR AUTOMATON PROGRAM FOR THE DEVELOPMENT OF FRENCH FLAG PATTERN. THE EVOLUTION WORKED WITH 10-INSTRUCTION PROGRAM, THE RESULTING SOLUTION CONTAINED 2 NOPS THAT WERE SUBSEQUENTLY REMOVED.

Line num.	Instruction
1:	MAX W C
2:	XOR C N
3:	MIN S E
4:	ROD
5:	AND E S
6:	DEC E
7:	OR C E
8:	XOR C W

V. CONCLUSIONS

In this paper we presented an instruction-based approach to the development of 2D cellular automata and their design using genetic algorithm. The idea was to shorten the genotype and reduce the search space especially for the CAs with more than 2 cell states. Two problems were considered in order to demonstrate the abilities of the proposed approach: (1) the replication problem and (2) the problem of development of a given pattern from a seed.

In case of the replication problem, the instruction-based approach overcame the conventional table-based transition function in all the performed experiments. We determined that in addition to the perfect success rate this method also reduces

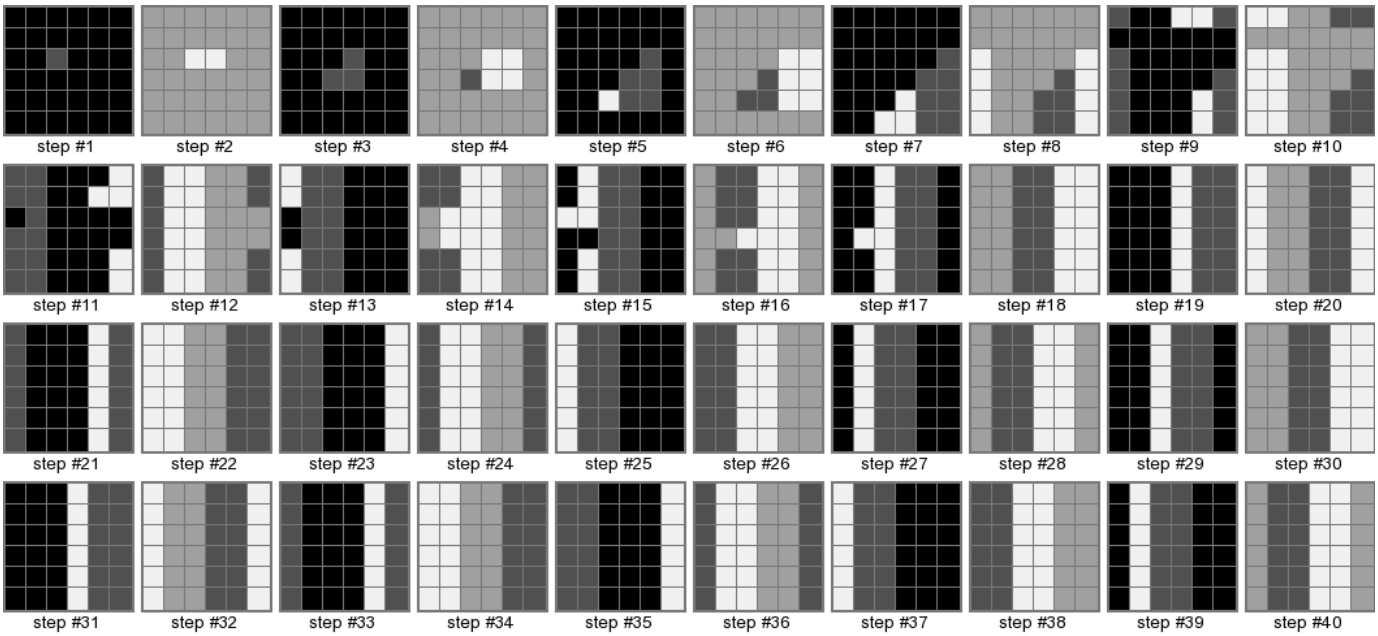

Fig. 7. Development of French flag pattern in a cellular automaton. This solution shows the development process in which the French flag emerges for the first time in step 26. Then the pattern is destroyed and emerges again with the period of 12 development steps (the next instance can be observed in step 38).

the computational effort needed to evolve a working solution of the replication problem.

The pattern development from a seed proposed interesting results in both of the instruction-based method and the conventional approach. Whilst the instruction-based development exhibits substantially higher success rate in most of the experiments, the conventional approach provides lover computational effort for obtaining a working solution.

In summary the proposed method works very well for more complex cellular automata, even for those in which no working solution was found by means of the conventional approach. We assume that the instruction-based approach is applicable to many other problems whose solution can be realized using cellular automata. The experiments that were performed in this paper represent problems for which successful solutions are known. However, we are going to experiment with more applications in order to determine the cellular automata behavior in different conditions. For example, the optimization of instruction set for a specific CA behavior seems to be an interesting area. Experiments in other application domains are in progress (e.g. development of computational structures or image operators may represent suitable candidates).

Acknowledgment

This work was supported by the Czech science foundation projects P103/10/1517 and GD102/09/H042, the research programme MSM 0021630528, the BUT projects FIT-S-11-1, FIT-S-12-1 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

[1] S. Wolfram, *A New Kind of Science*. Champaign IL: Wolfram Media, 2002.

[2] J. von Neumann, *The Theory of Self-Reproducing Automata*. A. W. Burks (ed.), University of Illinois Press, 1966.

[3] M. Sipper, *Evolution of Parallel Cellular Machines – The Cellular Programming Approach, Lecture Notes in Computer Science, volume 1194*. Berlin: Springer-Verlag, 1997.

[4] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, vol. 22, pp. 437–467, 1969.

[5] J. F. Miller, "Evolving developmental programs for adaptation, morphogenesis and self-repair," in *Advances in Artificial Life. 7th European Conference on Artificial Life, Lecture Notes in Artificial Intelligence, volume 2801*. Dortmund DE: Springer, 2003, pp. 256–265.

[6] T. Kowaliw, P. Grogono, and N. Kharma, "Bluenome: A novel developmental model of artificial morphogenesis," in *Proc. of the Genetic and Evolutionary Computation Conference, GECCO 2004, Lecture Notes in Computer Science, part I., volume 3102*. Springer-Verlag, 2004, pp. 93–104.

[7] P. C. Haddow and G. Tufte, "Bridging the genotype–phenotype mapping for digital FPGAs," in *Proc. of the 3rd NASA/DoD Workshop on Evolvable Hardware*. Los Alamitos, CA, US: IEEE Computer Society, 2001, pp. 109–115.

[8] 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.

[9] C. G. Langton, "Self-reproduction in cellular automata," *Physica D: Nonlinear Phenomena*, vol. 10, no. 1–2, pp. 135–144, 1984.

[10] Z. Pan and J. A. Reggia, "Computational discovery of instructionless self-replicating structures in cellular automata," *Artificial Life*, vol. 16, no. 1, pp. 39–63, 2010.

[11] M. Bidlo and J. Škarvada, "Instruction-based development: From evolution to generic structures of digital circuits," *International Journal of Knowledge-Based and Intelligent Engineering Systems*, vol. 12, no. 3, pp. 221–236, 2008.

[12] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.