Advances in the Evolution of Complex Cellular Automata

Michal Bidlo

Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Božetěchova 2, 61266 Brno, Czech Republic
E-mail: bidlom@fit.vutbr.cz
http://www.fit.vutbr.cz/~bidlom

Abstract. In this study we present some advanced experiments dealing with the evolutionary design of multi-state uniform cellular automata. The generic square calculation problem in one-dimensional automata will be treated as one of the case studies. An analysis of the evolutionary experiments will be proposed and properties of the resulting cellular automata will be discussed. It will be demonstrated that various approaches to the square calculations in cellular automata exist, some of which substantially overcome the known solution. The second case study deals with a non-trivial pattern development problem in two-dimensional automata. Some of the results will be presented which indicate that an exact behaviour can be automatically designed even for cellular automata working with more than ten cell states. A discussion for both case studies is included and potential areas of further research are highlighted.

Keywords: evolutionary algorithm, cellular automaton, transition function, conditional rule, square calculation, pattern development

1 Introduction

The concept of cellular automata was introduced by von Neumann in [15]. One of the aspects widely studied in his work was the problem of (universal) computational machines and the question about their ability to make copies of themselves (i.e. to self-reproduce). Von Neumann proposed a model with 29 cell states to perform this task. Later Codd proposed another approach and showed that the problem of computation and construction can be performed by means of a simplified model working with 8 states only [7].

Several other researchers studied cellular automata usually by means of various rigorous techniques. For instance, Sipper studied computational properties of *binary* cellular automata (i.e. those working with 2 cell states only) and proposed a concept of universal computing platform using a two-dimensional (2D) CA with non-uniform transition function (i.e. each cell can, in general, be controlled by a different set of transition rules) [21]. Sipper showed that, by introducing the non-uniform concept to the binary CAs, universal computation can be realised, which

was not possible using the Codd's model. In fact, Sipper's work significantly reduced the complexity of the CA in comparison with the models published earlier. Nevertheless even the binary uniform 2D CAs can be computationally universal if 9-cell neighbourhood is considered. Such CA was implemented using the famous rules of the Game of Life [2] (original proof of the concept was published in 1982 and several times revisited – e.g. see [8][11][17][18]).

Although binary CAs may be advantageous due to simple elementary rules and hardware implementations in particular, many operations and real-world problems can effectively be solved by *multi-state* cellular automata (i.e. those working with more than 2 cell states) rather than those using just two states. For example, a technique for the construction of computing systems in a 2D CA was demonstrated in [23] using rules of a simple game called Scintillae working with 6 cell states. Computational universality was also studied with respect to one-dimensional (1D) CA, e.g. in [13][27].

However, in some cases application specific operations (algorithms) may be more suitable than programming a universal system, allowing to better optimize various aspects of the design (e.g. resources, efficiency, data encoding etc.). For example, Tempesti [25] and Perrier et al. [12] showed that specific arrangements of cell states can encode sequences of instructions (programs) to perform a given operation. Wolfram presented various transition functions for CAs in order to compute elementary as well as advanced functions (e.g. parity, square, or prime number generation) [26]. Further problems were investigated in recent years [16][19].

In addition to the computational tasks, various other (more geneal) benchmark problems have been investigated using cellular automata, e.g. including principles of self-organization, replication or pattern formation. For example, Basanta et al. used a genetic algorithm to evolve the rules of effector automata (a generalised variant of CA) to create microstructural patterns that are similar to crystal structures known from some materials [1]. An important aspect of this work was to investigate new materials with specific properties and their simulation using computers. Suzudo proposed an approach to the evolutionary design of 2D asynchronous CA for a specific formation of patterns in groups in order to better understand of the pattern-forming processes known from nature [24]. Elmenreich et al. proposed an original technique for growing self-organising structures in CA whose development is controlled by neural networks according to the internal cell states [9].

The proposed work represents an extended version of our recent study published in [4], the aim of which is to present a part of our wider research in the area of cellular automata, where representation techniques and automatic (evolutionary) methods for the design of complex multi-state cellular automata are investigated. The goal of this work is to design transition functions for cellular automata using evolutionary algorithms, which satisfy the given behaviour with respect to some specific initial and target conditions. In particular, it will be shown that the evolutionary algorithm can design various transition functions for uniform 1D CAs (that have never been seen before) to perform *generic*

square calculations in the cellular space using just local interactions of cells. An additional analysis of the results demonstrates that various generic CA-based solutions of the squaring problem can be discovered, which substantially overcome the known solution regarding both the complexity of the transition functions and the number of steps (speed) of calculation. In order to show the abilities of the proposed method for designing CA using the concept of conditionally matching rules, some further experiments are presented regarding the evolution of 2D multi-state cellular automata in which the formation of some non-trivial patterns is treated as a case study. As cellular automata represent a platform potentially important for future technologies (see their utilisation in various emerging fields, e.g. [14], [22] or [20]), it is worth studying their design and behaviour on the elementary level as well (i.e. using various benchmark problems).

2 Cellular Automata for Square Calculations

For the purposes of developing algorithms for squaring natural numbers, 1D uniform cellular automata are treated with the following specification (target behaviour). The number of cell states is investigated for values 4, 6, 8 and 10 (this was chosen on the basis of the existing solution [26] that uses 8 states; moreover it is worth of determining whether less states will enable to design generic solutions and whether the EA will be able to find solutions in a huge search space induced by 10 cell states). The new state of a given cell depends on the states of its west neighbour (c_W), the cell itself (central cell, c_C) and its east neighbour (c_E), i.e. it is a case of 3-cell neighbourhood. *A step* of the CA will be considered as a synchronous update of state values of all its cells according to a given transition function. For the practical implementation purposes, cyclic boundary conditions are considered. However, it is important to note that CAs with sufficient sizes are used in order to avoid affecting the development by the finite number of cells.

The value of x is encoded in the initial CA state as a continuous sequence of cells in state 1, whose length (i.e. the number of cells in state 1) corresponds to x, the other cells possess state 0. For example, the state of a 12-cell CA, which encodes $x = 3$, can appear as 0000011100000. The result $y = x^2$, that will emerge from the initial state in a finite number of steps, is assumed as a stable state in which a continuous sequence of cells in non-zero states can be detected, the length of which equals the value of y, the other cells are required in state 0. For the aforementioned example, the result can appear as 002222222220 or even 023231323200 (there is a sequence of non-zero cells of length $3^2 = 9$). The concept of representing the input value x and the result y is graphically illustrated in Figure 1. This is a generalised interpretation based on the idea presented in [26], page 639. The goal is to discover transition functions for the CA, that are able to calculate the square of arbitrary number $x > 1$.

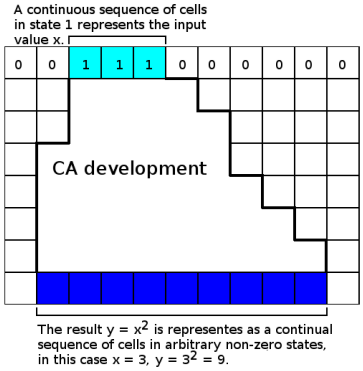


Fig. 1. Illustration of encoding integer values in a 1D cellular automaton. In this example $x = 3, y = 9$. Extracted from [4].

2.1 Conditionally Matching Rules

In order to represent the transition functions for CAs, the concept of Conditionally Matching Rules (CMR), originally introduced in [6], will be applied. This technique showed as very promising for designing complex cellular automata [3][5]. For the 1D CA working with 3-cell neighbourhood, a CMR is defined as $(cond_W\ s_W)(cond_C\ s_C)(cond_E\ s_E) \rightarrow s_{Cnew}$, where $cond_\bigstar$ denotes a condition function and $s_\bigstar$ denotes a state value. Each part $(cond_\bigstar\ s_\bigstar)$ on the left of the arrow is evaluated with respect to the state of a specific cell in the neighbourhood (in this case c_W, c_C and c_E respectively). For the experiments presented in this work the relation operators $=$, $\neq$, $\geq$ and $\leq$ are considered as the condition functions. A finite sequence of CMRs represents a transition function. In order to determine the new state of a cell, the CMRs are evaluated sequentially. If a rule is found in which all conditions are true (with respect to the states in the cell neighbourhood), s_{Cnew} from this rule is the new state of the central cell. Otherwise the cell state does not change. For example, consider a transition function that contains a CMR $(\neq 1)(\neq 2)(\leq 1) \rightarrow 1$. Let c_W, c_C, c_E be states of cells in a neighbourhood with values $2, 3, 0$ respectively, and a new state of the central cell ought to be calculated. According to the aforementioned rule, $c_W \neq s_W$ is true as $2 \neq 1$, similarly $c_C \neq s_C$ is true $(3 \neq 2)$ and $c_E \leq s_E$ $(0 \leq 1)$. Therefore, this CMR is said to match, i.e. $s_{Cnew} = 1$ on its right side will update the state of the central cell. Note that the same concept can also be applied to CA working with a wider cellular neighborhood. For example, a CMR for a 2D CA with 5-cell neighborhood would consist of 5 items for the conditional functions instead for 3 items.

The evolved CMRs can be transformed to the conventional table rules [5] without loss of functionality or violating the basic CA principles. In this work the transformation is performed as follows: (1) For every possible combination of states $c_W\ c_C\ c_E$ in cellular neighborhood a new state s_{Cnew} is calculated using the CMR-based transition function. (2) If $c_C \neq s_{Cnew}$ (i.e. the cell state ought

to be modified), then a table rule of the form $c_W\ c_C\ c_E \rightarrow s_{Cnew}$ is generated. Note that the combinations of states not included amongst the table rules do not change the state of the central cell, which is treated implicitly during the CA simulation. The number of such *generated rules* will represent a metrics indicating the complexity of the transition function.

In order to determine the complexity of the transition function with respect to a specific square calculation in CA, a set of *used rules* is created using the aforementioned principle whereas the combinations of states $c_W\ c_C\ c_E$ are considered just occurring during the given square calculation in the CA. There metrics (together with the number of states and CA steps) will allow us to compare the solutions obtained by the evolution and to identify the best results with respect to their complexity and efficiency.

An evolutionary algorithm will be applied to search for suitable CMR-based transition functions as described in the following section.

3 Setup of the Evolutionary System

A custom evolutionary algorithm (EA) was utilised, which is a result of our long-term experimentation in this area. Note, however, that neither tuning of the EA nor in-depth analysis of the evolutionary process is a subject of this work. The EA is based on a simple genetic algorithm [10] with a tournament selection of base 4 and a custom mutation operator. Crossover is not used as it has not shown any improvement in success rate or efficiency of our experiments.

The EA utilises the following fixed-length representation of the conditionally matching rules in the genomes. For the purpose of encoding the condition functions $=$, $\neq$, $\geq$ and $\leq$, integer values 0, 1, 2 and 3 will be used respectively. Each part $(cond_\bigstar\ s_\bigstar)$ of the CMR is encoded as a single integer $P_\bigstar$ in the range from 0 to M where $M = 4 * S - 1$ (4 is the fixed number of condition functions considered and S is the number of cell states) and the part $\rightarrow s_{Cnew}$ is represented by an integer in the range from 0 to $S - 1$. In order to decode a specific condition and state value, the following operations are performed: $cond_\bigstar = P_\bigstar / S$, $s_\bigstar = P_\bigstar\ mod\ S$ (note that / is the integer division and *mod* is the modulo-division). This means that a CMR $(cond_W\ s_W)(cond_C\ s_C)(cond_E\ s_E) \rightarrow s_{Cnew}$ can be represented by 4 integers; if 20 CMRs ought to be encoded in the genome, then $4 * 20 = 80$ integers are needed. For example, consider $S = 3$ for which $M = 4 * 3 - 1 = 11$. If a 4-tuple of integers (2 9 11 2) representing a CMR in the genome ought to be decoded, then the integers are processed respectively as:

- $cond_W = 2/3 = 0$ which corresponds to the operator $=$, $s_W = 2\ mod\ 3 = 2$,
- $cond_C = 9/3 = 3$ which corresponds to the operator $\leq$, $s_C = 9\ mod\ 3 = 0$,
- $cond_E = 11/3 = 3$ which corresponds to the operator $\leq$, $s_E = 11\ mod\ 3 = 2$,
- $s_{Cnew} = 2$ is directly represented by the 4th integer.

Therefore, a CMR of the form $(=\ 2)(\leq\ 0)(\leq\ 2) \rightarrow 2$ has been decoded.

The following variants of the fitness functions are treated (note that the input x is set to the middle of the cellular array):

1. RESULT ANYWHERE (RA-fitness): The fitness is calculated with respect to any valid arrangement (position) of the result sequence in the CA. For example, $y = 4$ in an 8-cell CA may be $rrrr0000$, $0rrrr000$, $00rrrr00$, $000rrrr0$ or $0000rrrr$, where $r \neq 0$ represent the result states that may be generally different within the result sequence. A partial fitness value is calculated for every possible arrangement of the result sequence as the sum of the number of cells in the expected state for the given values of x. The final fitness is the highest of the partial fitness values.
2. SYMMETRIC RESULT (SR-fitness): The result is expected symmetrically with respect to the input. For example, if 0000011100000 corresponds to initial CA state for $x = 3$, then the result $y = 3^2$ is expected as a specific CA state $00rrrrrrrrr00$ (each r may be represented by any non-zero state). The fitness is the number of cells in the expected state.

The fitness evaluation of each genome is performed by simulating the CA for initial states with the values of x from 2 to 6. The result of the x^2 calculation is inspected after the 99th and 100th step of the CA, which allows to involve the state stability check into the evaluation. This approach was chosen on the basis of the maximal x evaluated during the fitness calculation and on the basis of the number of steps needed for the square calculation using the existing solution [26]. In particular, the fitness of a fully working solution evaluated for x from 2 to 6 in a 100-cell CA is given by $F_{max} = 5 * 2 * 100 = 1000$ (there are 5 different values of x for which the result x^2 is investigated in 2 successive CA states, each consisting of 100 cells). The evolved transition functions, satisfying the maximal fitness for the given range of x, are checked for the ability to work in larger CAs for up to $x = 25$ The solutions which pass this check are considered as generic.

The EA works with a population of 8 genomes initialised randomly at the beginning of evolution. After evaluating the genomes, four candidates are selected randomly, the candidate with the highest fitness becomes a parent. An offspring is created by mutating 2 randomly selected integers in the parent. The selection and mutation continue until a new population of the same size is created and the evolutionary process is repeated until 2 million generations are performed. If a solution with the maximal fitness is found, then the evolutionary run is considered as successful. If no such solution is found within the given generation limit, then the evolutionary run is terminated and regarded as unsuccessful.

4 Results of Square Calculations in 1D CA

The evolutionary design of CAs for the generic square calculation has been investigated for the following settings: the number of states 4, 6, 8 and 10, the transition functions consisting of 20, 30, 40 and 50 CMRs and two ways of the fitness calculation described in Section 3. For each setup, 100 independent evolutionary runs have been executed. The success rate and average number of generations needed to find a working solution were observed with respect to the evolutionary process. As regards the parameters of the CA, the minimal number of rules and steps needed to calculate the square of x were determined.

Table 1. Statistics of the evolutionary experiments conducted using the RA-fitness (the upper part of the table) and the parameters of the generic solutions (in the lower part of the table). The parameters of the best results obtained are marked **bold**. Note that # denotes "the number of", the meaning of "generated rules", "used rules" and "steps" of the CA is defined in Section 2. Extracted from [4].

	the number of states															
	4				6				8				10			
the num. of CMRs	succ. rate	avg. gen.	min. steps	min. rules	succ. rate	avg. gen.	min. steps	min. rules	succ. rate	avg. gen.	min. steps	min. rules	succ. rate	avg. gen.	min. steps	min. rules
20	3	844364	54	35	30	769440	45	120	45	570939	39	232	35	328210	47	569
30	3	620998	52	36	24	749837	40	120	38	595467	42	340	33	363360	45	663
40	2	1344286	77	46	19	629122	37	136	30	701612	41	365	29	244566	46	662
50	2	959689	73	43	20	813803	41	134	35	582342	39	348	38	373490	40	762
	the number of generic solutions (#generic) obtained for the given number of states and parameters of the generic solutions: #generated rules/#used rules/#steps for 6^2)															
#generic, parameters	1 **36/26/74**				5 **176/52/46**, 164/33/87, 152/49/78, 185/66/70, 175/52/69				6 435/49/68, 403/51/79, 422/39/65, 392/62/76, 423/41/68, 429/94/76				3 934/64/56, 835/61/79, 916/35/76			

For the purposes of comparison of the results proposed in Section 4.3, the CA will be denominated by unique identifiers of the form CA–XX–YY, where XX and YY are integers distinguishing the sets of evolutionary experiments and the CA obtained.

4.1 Results for the RA-Fitness

For the RA-fitness, the statistical results are summarised in Table 1. The table also contains the total numbers of generic solutions discovered for the given state setups and parameters determined for these solutions. For every number of states considered, at least one generic solution was identified. For example, a transition function was discovered for the 4-state CA, which consists of 36 table rules (transformed from the CMR representation evolved). This solution can be optimised to 26 rules (by eliminating the rules not used during the square calculation) which represents the simplest CA for generic square calculations known so far (note that Wolfram's CA works with 8 states and 51 rules [26]). Moreover, for example, our solution needs 74 steps to calculate 6^2 whilst Wolfram's CA needs 112 steps, which also represents a substantial innovation discovered by the EA. The CA development corresponding to this solution is shown in Figure 2.

Another result obtained using the RA-fitness is illustrated by the CA development in Figure 3. In this case the CA works with 6 states and its transition function consists of 52 effective rules. The number of steps needed, for example, to calculate 6^2, is 46 (and compared to 112 steps of Wolframs CA, it is an improvement of the CA efficiency by more than 50%) which represents the best CA known so far for this operation and the best result obtained from our experiment.

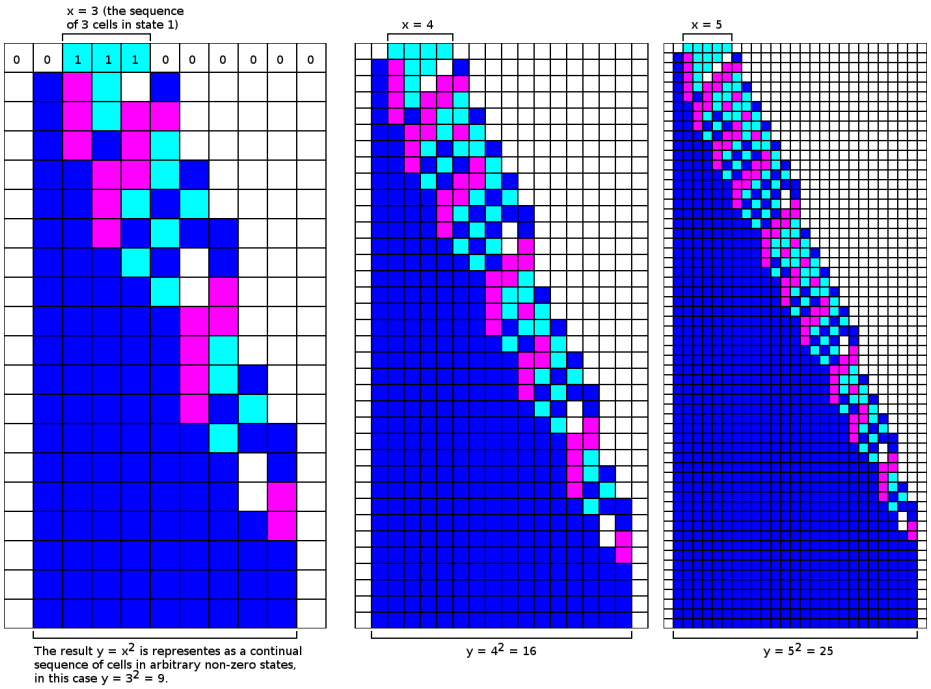


Fig. 2. Example of a 4-state squaring CA development for $x = 3, 4$ and 5 using our most compact transition function. This solution is denominated as CA–30–00 and its rules are: 0 0 1 → 3, 0 1 1 → 2, 0 3 0 → 2, 1 0 0 → 3, 1 0 2 → 2, 1 0 3 → 2, 1 1 0 → 0, 1 1 2 → 2, 1 1 3 → 2, 1 2 1 → 1, 1 3 0 → 1, 1 3 1 → 1, 1 3 2 → 2, 1 3 3 → 0, 2 1 2 → 3, 2 1 3 → 3, 2 2 0 → 1, 2 2 1 → 1, 2 3 1 → 1, 2 3 2 → 2, 3 0 2 → 3, 3 1 0 → 3, 3 1 1 → 3, 3 1 3 → 3, 3 2 0 → 3, 3 2 3 → 3. Extracted from [4].

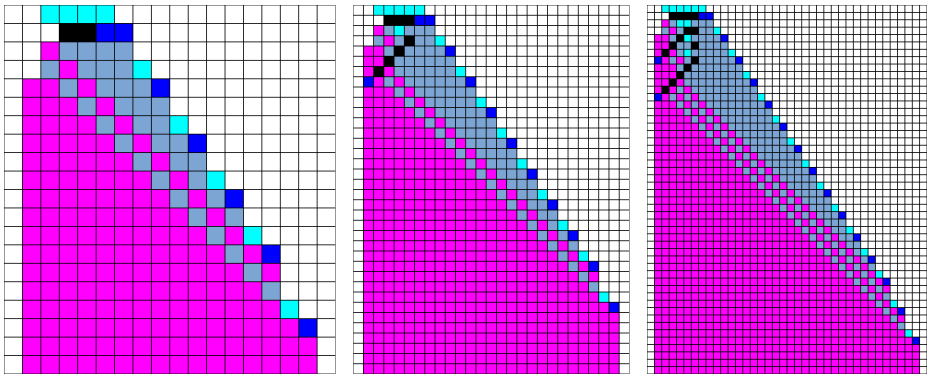


Fig. 3. Example of a 6-state CA development for $x = 4, 5$ and 6. This is the fastest CA-based (3-neighbourhood) solution known so far and the best result obtained from our experiments. This solution is denominated as CA–50–12 and its rules are: 0 0 4 → 2, 0 0 5 → 2, 0 1 1 → 0, 0 2 3 → 0, 0 2 4 → 4, 0 2 5 → 3, 0 3 2 → 2, 0 3 3 → 0, 0 4 0 → 2, 0 4 2 → 2, 0 5 3 → 0, 0 5 5 → 4, 1 0 0 → 3, 1 1 0 → 3, 1 1 1 → 5, 1 1 4 → 5, 1 4 4 → 5, 2 0 4 → 3, 2 1 0 → 2, 2 2 5 → 5, 2 3 0 → 2, 2 3 4 → 2, 2 4 0 → 2, 2 4 1 → 2, 2 4 2 → 2, 2 4 3 → 2, 2 4 4 → 2, 2 4 5 → 5, 2 5 2 → 2, 2 5 4 → 2, 3 3 0 → 4, 3 4 4 → 2, 4 0 0 → 1, 4 1 0 → 4, 4 1 1 → 4, 4 1 4 → 4, 4 2 0 → 4, 4 2 1 → 4, 4 2 3 → 4, 4 2 4 → 4, 4 3 0 → 4, 4 4 5 → 1, 4 5 1 → 4, 4 5 4 → 4, 4 5 5 → 1, 5 1 1 → 4, 5 1 4 → 4, 5 2 4 → 4, 5 3 3 → 4, 5 5 3 → 4, 5 5 4 → 4, 5 5 5 → 1. Extracted from [4].

One more example of evolved CA is shown in Figure 4. This generic solution was obtained in the setup with 8-state CA, however, the transition function works with 6 different states only. There are 49 transition rules, the CA needs 68 steps to calculate 6^2. This means that the EA discovered a simpler solution (regarding the the number of states and table rules) which is a part of the solution space of the 8-state CA. Again, this result exhibits generally better parameters compared to the known solution from [26]. The CA development, that was not observed in any other solution, is also interesting visually - as Fig. 4 shows, the CA generates a pattern with some "dead areas" (cells in state 0) within the cells that subsequently form the result sequence. The size of these areas is gradually reduced, which finally lead to derive the number of steps after which a stable state containing the correct result for the given x has emerged (illustrated by the right part of Figure 4 for $x = 8$ whereas the CA needs 122 steps to produce the result).

4.2 Results for the SR-Fitness

Table 2 shows the statistics for the SR-fitness together with the total numbers of generic solutions discovered for the given state setups and parameters determined for these solutions. As evident, the success rates are generally lower compared to the RA-fitness which is expectable because the SR-fitness allows a single arrangement only of the result sequence in the CA. Moreover, just two generic

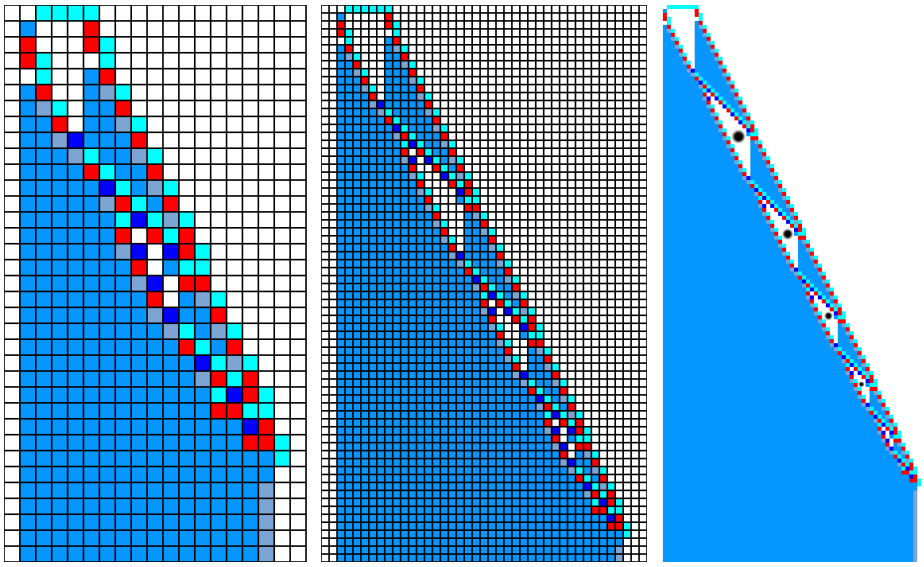


Fig. 4. Example of a 6-state squaring CA development (originally designed using 8-state setup) for $x = 4, 6$ and 8. This solution is denominated as CA–40–01. Its development shows a specific pattern evolved to derive the result of x^2, which was not observed in any other solution. The part on the right shows a complete global behaviour of this CA for $x = 8$ with some "dead areas" (marked by black spots) which lead to the correct stable result by progressively reducing the size of these areas (in this case the result of 8^2 is achieved after 122 steps). Extracted from [4].

CAs have been identified out of all the runs executed for this setup. However, the goal of this experiment was rather to determine whether solutions of this type ever exist for cellular automata and evaluate the ability of the EA to find them. As regards both generic solutions, their numbers of used rules and CA steps are significantly better in comparison with Wolfram's solution [26]. Specifically, Wolfram's solution uses are 51 rules and the calculation of 6^2 takes 112 steps, whilst the proposed results use 33, respective 36 rules and calculate 6^2 in 71, respective 78 steps. Moreover, one of them was discovered using a 4-state CA (Wolfram used 8 states), which belongs to the most compact solutions obtained herein and known so far.

Table 2. Statistics of the evolutionary experiments conducted using the SR-fitness (the upper part of the table) and the parameters of the generic solutions (in the lower part of the table). The parameters of the best result obtained are marked **bold**. Note that # denotes "the number of", the meaning of "generated rules", "used rules" and "steps" of the CAs is defined in Section 2. Extracted from [4].

	the number of states															
	4				6				8				10			
the num. of CMRs	succ. rate	avg. gen.	min. steps	min. rules	succ. rate	avg. gen.	min. steps	min. rules	succ. rate	avg. gen.	min. steps	min. rules	succ. rate	avg. gen.	min. steps	min. rules
20	2	634948	71	38	4	734200	38	126	11	982446	34	234	18	855791	53	542
30	0	-	-	-	5	905278	48	150	17	934123	51	327	15	910269	35	742
40	1	1546681	79	45	4	928170	33	147	11	1033059	53	317	15	898314	52	748
50	0	-	-	-	3	989039	44	138	12	811686	32	380	17	861850	52	796
	the number of generic solutions (#generic) obtained for the given number of states and parameters of the generic solutions: #generated rules/#used rules/#steps for 6^2)															
#generic, parameters	1 **38/33/71**				0				1 234/36/78				0			

Figure 5 shows examples of a CA (identified as generic) evolved using the SR-fitness. The transition function, originally obtained in 8-state CA setup, is represented by 36 used rules and works with 7 states only. Although this result cannot be considered as very efficient (for 6^2 the CA needs 78 steps), it exhibits one of the most complex emergent process obtained for the square calculation, the result of which is represented by a non-homogeneous state. The sample on the right of Fig. 5 shows a cutout of development for $x = 11$ in which the global behaviour can be observed. This result demonstrates that the EA can produce generic solutions to a non-trivial problem even for a single specific position of the result sequence required by the SR-fitness evaluation.

4.3 Analysis and Comparison of the Results

In this section an overall analysis and comparison of the results obtained for the generic square calculations is provided and some of further interesting CA are shown. Note that the data related to the CA behavior and visual samples

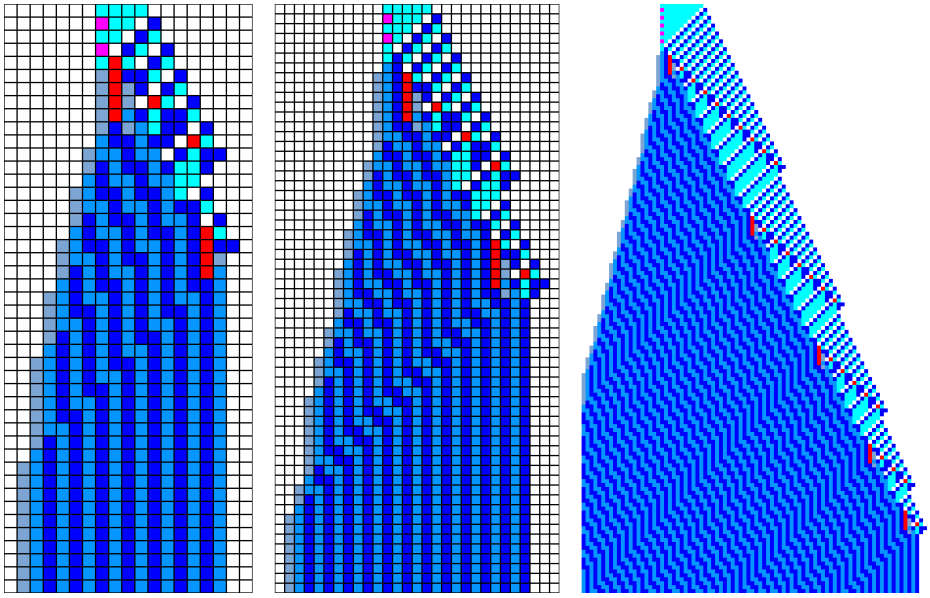


Fig. 5. Example of a 7-state CA controlled by a transition function evolved using the SR-fitness. This solution is denominated as CA–20–07. A complete development is shown for $x = 4$ and 5 (the left and middle sample respectively), the part on the right demonstrates a cutout of global behaviour of the CA for $x = 11$. Extracted from [4].

of selected calculations were obtained using our experimental software (i.e. the evolutionary system and a dedicated CA simulator developed specifically for this purpose). There are in total 17 different CA obtained from our experiments and included in this evaluation.

In order to provide a direct comparison of computational efficiency of the CA, the number of steps needed to calculate the square was evaluated for the values of the input integer x from 2 to 16. We used the WolframAlpha computational knowledge engine[1] to generate expressions which allow us to determine the number of steps of a given CA for $x > 16$. Tables 3 and 4 summarize the analysis. The resulting CA are sorted from the best to worst regarding the number of steps for $x = 16$ as the sorting criterion.

For some CA the equations derived by WolframAlpha are specified for an independent integer variable $n > 0$ by means of which the number of steps can be determined for a given input value of x. For example, the number of steps a_n of the CA–50–27 from Table 3 can be determined as $a_n = n(2n + 3)$ for all $x \geq 1$, i.e. in order to calculate the number of steps for $x = 7$, for example, then $n = x - 1 = 6$ and the substitution of this value to the equation gives $a_n = 6 * (2 * 6 + 3) = 90$ steps which corresponds to the value from Table 3 for $x = 7$ (observed for this CA using our CA simulator). The reason for taking $n = x - 1$ follows from the fact that the cellular automata work for $x \geq 2$.

For some CA WolframAlpha derived an iterative expression determining the number of steps a_{n+1} from the previous value a_n. For example, the CA–50–12 (the best one in this work) allows determining the number of steps for a given x as $a_{n+1} = n(3n + 7) - a_n$ for $n \geq 2$. Therefore, in order to calculate the number of steps e.g. for $x = 8$, it is needed to take $n = x - 2 = 8 - 2 = 6$, the value for the previous $x = 7$ must be known, i.e. $a_n = 64$ from Table 3, and substituting to the equation $a_{n+1} = 6 * (3 * 6 + 7) - 64 = 86$ which is the number of CA steps needed to calculate the result of 8^2 (as corresponds to the value from Table 3 for this CA for $x = 8$ observed in our CA simulator).

The aforementioned example also shows that it is not possible in some cases to express the number of CA steps for arbitrary $n \geq 1$ which means that the development of some CA for low values of x exhibit some anomaly in comparison with the development for larger input values. The reason for this behavior still remains in general an open question but our observations indicate that this is probably the case of CA exhibiting a complex (and mostly visually very attractive) pattern generated by the development. For a low input value (e.g. $x < 5$ in case of the CA–40–14 from Table 3) the development does not need to involve all possible state transitions before reaching the result state, which would otherwise emerge for larger x. Therefore, the development for $x < 5$ is specific, leading to the numbers of steps that is not possible to express together with the numbers of steps for larger input values.

The overall comparison of some selected CA is shown in Figure 6. As evident from this figure (and from the equations in Table 3 and 4), the computational efficiency of the CA (i.e. the number of steps needed for given x) in all cases

[1] https://www.wolframalpha.com/

Table 3. The number of steps of resulting CA needed to calculate the square for $x = 2, ..., 16$ together with expressions allowing to calculate the number of steps for given $n = x - 1$. (Part 1 of the CA comparison.)

the input value of x														
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
the equation derived for the num. of steps of a given CA and the num. of CA steps needed to finish the calculation of x^2														
CA–50–12: $a_{n+1} = n(3n+7) - a_n$ for $n \geq 2$														
5	11	18	30	46	64	86	110	138	168	202	238	278	320	366
CA–30–08: $a_n = 2n^2 + n + 1$, $n \geq 1$														
4	11	22	37	56	79	106	137	172	211	254	301	352	407	466
CA–50–27: $a_n = n(2n+3)$, $n \geq 1$														
5	14	27	44	65	90	119	152	189	230	275	324	377	434	495
CA–40–01: $a_n = 2n^2 + 3n + 3$, $n \geq 1$														
8	17	30	47	68	93	122	155	192	233	278	327	380	437	498
CA–40–14: $a_{n+1} = \frac{a_n(n-3)}{n-5} - \frac{2(11n+13)}{n-5}$, $n \geq 5$														
8	16	29	46	68	94	124	158	196	238	284	334	388	446	508
CA–30–11: $a_{n+1} = -a_n + 4n^2 + 13n + 11$ for $n \geq 2$														
3	17	35	51	76	100	133	165	206	246	295	343	400	456	521
CA–30–00: $a_{n+1} = \frac{a_n(n-1)}{n-3} + \frac{-15n-19}{n-3}$, $n \geq 3$														
4	17	32	51	74	101	132	167	206	249	296	347	402	461	524
CA–50–22: $a_{n+1} = 2(2n^2 + 7n + 7) - a_n$ for $n \geq 1$														
14	24	34	58	76	108	134	174	208	256	298	354	404	468	526
CA–20–07: $a_n = \frac{1}{4}(8n^2 + 22n - 5(-1)^n - 3)$, $n \geq 1$														
8	17	35	52	78	103	137	170	212	253	303	352	410	467	533

Table 4. The number of steps of resulting CA needed to calculate the square for $x = 2, ..., 16$ together with expressions allowing to calculate the number of steps for given $n = x - 1$. (Part 2 of the CA comparison.)

the input value of x														
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
the equation derived for the num. of steps of a given CA and the num. of CA steps needed to finish the calculation of x^2														
CA–40–34: $a_n = \frac{1}{4}(8n^2 + 22n - 3(-1)^n + 3),\ n \geq 1$														
9	19	36	54	79	105	138	172	213	255	304	354	411	469	534
CA–50–07: $a_n = \frac{1}{2}(5n^2 + 3n - 2),\ n \geq 1$														
3	12	26	45	69	98	132	171	215	264	318	377	441	510	584
CA–20–00: $a_{n+1} = -a_n + 5n^2 + 9n + 2$ for $n \geq 1$														
5	13	27	47	71	101	135	175	219	269	323	383	447	517	591
CA–30–31: $a_{n+1} = \frac{a_n(n^2-8n-2)}{n^2-10n+7} + \frac{-27n^2+13n+9}{n^2-10n+7},\ n \geq 9$														
2	16	33	54	79	108	141	178	219	271	331	397	469	547	631
CA–50–17: $a_n = 3n^2 + 1,\ n \geq 1$														
4	13	28	49	76	109	148	193	244	301	364	433	508	589	676
CA–20–27: $a_{n+1} = \frac{a_n(n-1)}{n-3} - \frac{15(n+1)}{n-3},\ n \geq 3$														
7	15	30	51	78	111	150	195	246	303	366	435	510	591	678
CA–50–15: $a_n = 3n^2 + 2n + 2,\ n \geq 1$														
7	18	35	58	87	122	163	210	263	322	387	458	535	618	707
Wolfram's CA: $a_n = 3n^2 + 7n + 2,\ n \geq 1$														
12	28	50	78	112	152	198	250	308	372	442	518	600	688	782

exhibit a quadratic form. However, the CA efficiency differ substantially between various solutions. Whilst Wolfram's CA exhibits the highest numbers of steps out of all CA that were available for the square calculations herein, our experiments showed that (1) this approach can be improved substantially and (2) various other solutions with a moderate efficiency exist for this task. Some of them are presented as visualisation of the appropriate CA development in Figure 7 and 8. Specifically, the CA–30–08 from Figure 7a represents an example of the second best result discovered from our evolutionary experiments. Its simple pattern exhibit a high degree of regularity which is probably the cause of very simple equation expressing the number of steps for given x (Table 3). More complex, less computationally efficient and visually interesting patterns are generated by CA–50–22 (Figure 7b, Table 3) and CA–40–34 (Figure 7c, Table 4). On the other hand, the CA–30–31 from Figure 8a produces results of calculating x^2 that is composed of various (stable) state values and this CA also exhibit the most complex equation needed to express its number of steps for given x (see Table 4). Together with CA from Figure 8b,c it belongs to the least computationally efficient (i.e. requires many steps to produce the result – see the comparison in Figure 6) but also can be viewed as solutions that demonstrate the variety of different styles of how the result of x^2 in CA can be achieved.

4.4 Discussion

In most cases of the experimental settings the EA was able to produce at least one generic solution for the CA-based square calculation. Despite the 2 million generation limit, the results from Table 1 and 2 show that the average number of generations is mostly below 1 million, which indicates a potential of the EA to efficiently explore the search space. In comparison with the initial study of this problem proposed in [5], where 200,000 generations were performed, the significant increase of this parameter herein is important with respect to achieving a reasonable success rate and producing generic solutions (note that an initial comparison of various ranges for x evaluated in the fitness was proposed in [5], the result of which was considered in this work).

As regards the RA-fitness, which can be considered as the main technique proposed herein for the evolution of cellular automata, a more detailed analysis was performed with various multi-state CA. As the results in Table 1 show that the number of generic solutions increases for the number of states from 4 to 8, then for 10-state CAs a significant reduction can be observed. This is probably caused by the exponential increase of the search space depending on the number of states. The results indicate that the 8-state setup represents a very feasible value that may be considered as sufficient for this kind of problem (note that 6 generic solutions were obtained for this setup).

In both sets of experiments with the RA-fitness and SR-fitness, a phenomenon of a reduction of the number of states was observed. This is possible due to the identification of just the rules that are needed for the CA development to calculate the square out of all the rules generated from the evolved CMR-based

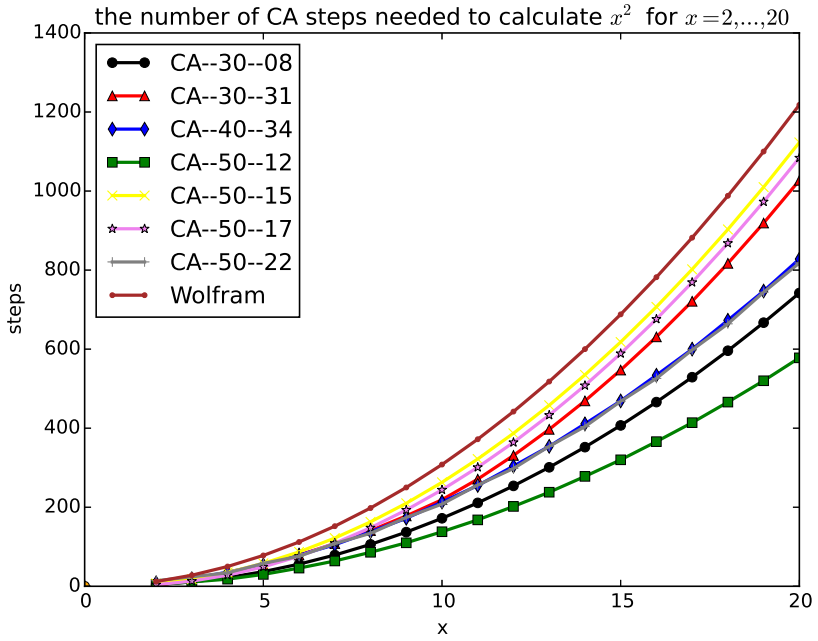
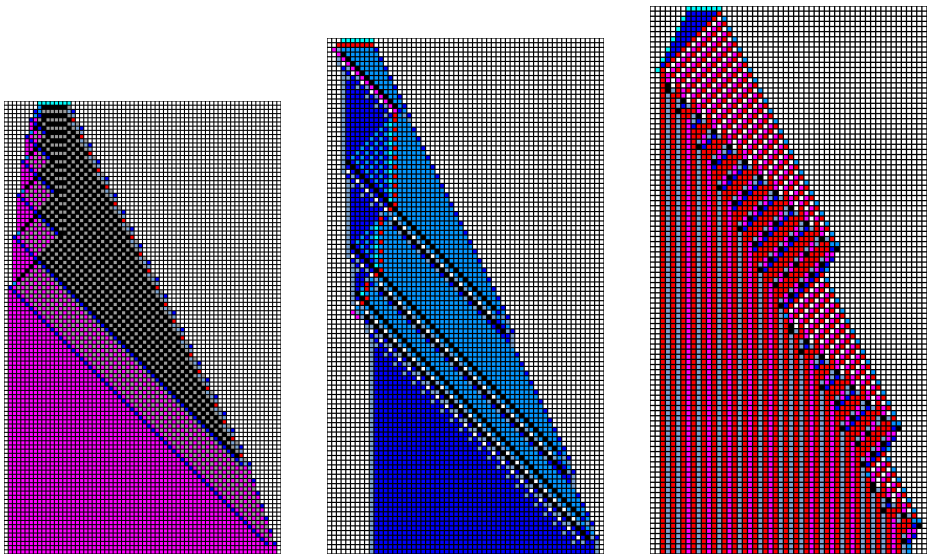


Fig. 6. Evaluation of the computational efficiency (i.e. the number of steps needed to achieve the result of x^2) of some selected CA whose development is also presented visually in various figures in this paper. A comparison with the existing Wolfram's CA [26] (the top-most curve) is included.



(a) (b) (c)

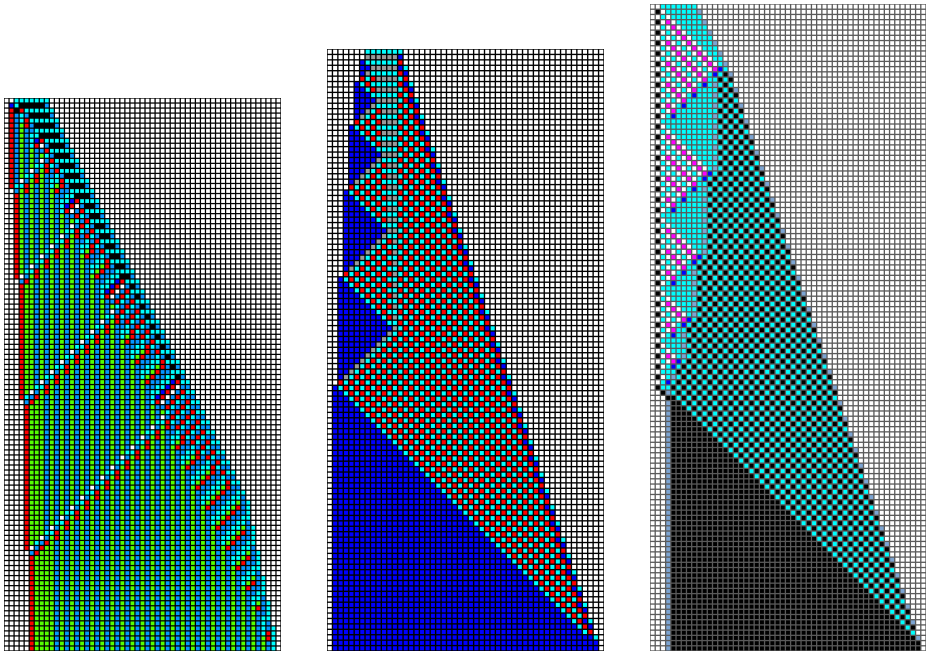
Fig. 7. Visualisation of the development of some selected squaring cellular automata obtained from our experiment: (a) CA–30–08, $x = 8$, (b) CA–50–22, $x = 7$, (c) CA–40–34, $x = 7$.

transition function for every valid combination of states in the cellular neighbourhood. It was determined that the CAs in some cases do not need all the available cell states to perform the given operation.

5 Evolution of Complex 2D Cellular Automata

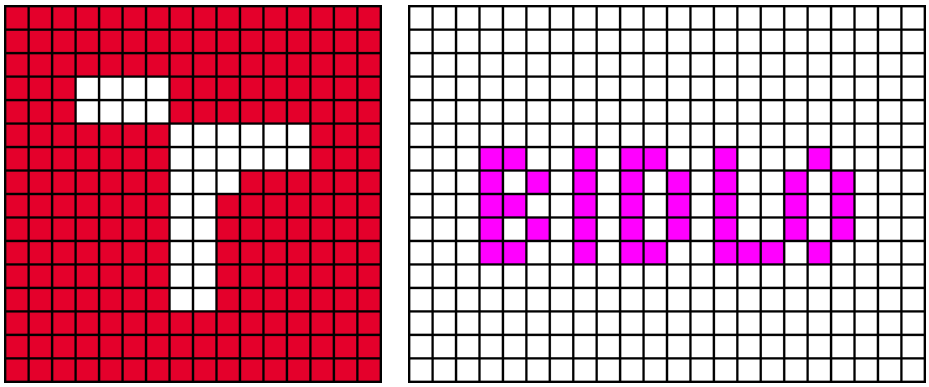
In order to provide a wider overview of what CA the proposed method can handle, some experiments dealing with the evolution of uniform multi-state 2D automata were conducted. The pattern development problem was chosen as a case study. In particular, some non-trivial and asymmetric patterns were chosen as shown in Figure 9.

In order to evaluate a candidate CA, up to 40 development steps were performed and the steps 16-40 was assigned a partial fitness calculated as the number of cells in correct states with respect to the given pattern. The final fitness of the candidate CA was the maximum of the partial fitness values. The reason for this setup is to reduce the time needed for the evaluation because the patterns probably cannot be finished in less than 16 steps (this values was determined empirically). Therefore, no inspection of cell states is performed in steps 1-15. Moreover, no exact number of steps is known in which a pattern can be finished so that the aforementioned range of steps provides the evolution with a wider



(a) (b) (c)

Fig. 8. Visualisation of the development of some selected squaring cellular automata obtained from our experiment: (a) CA–30–31, $x = 7$, (b) CA–50–17, $x = 7$, (c) CA–50–15, $x = 7$.



(a) (b)

Fig. 9. Samples of selected patterns treated in the experiments for the pattern development problem in 2D CA: (a) the BUT logo (note that the T-like structure, composed of cells in state 1, is the subject of the CA development on the red background – cells in state 0), (b) a label containing the author's surname (composed of cells in state 1, the other cells are in state 0).

space to discover a solution. This setup does not eliminate a possibility of destroying the pattern that emerged at a certain step during the subsequent CA development. However, the development of a stable pattern was not a primary goal of these experiments.

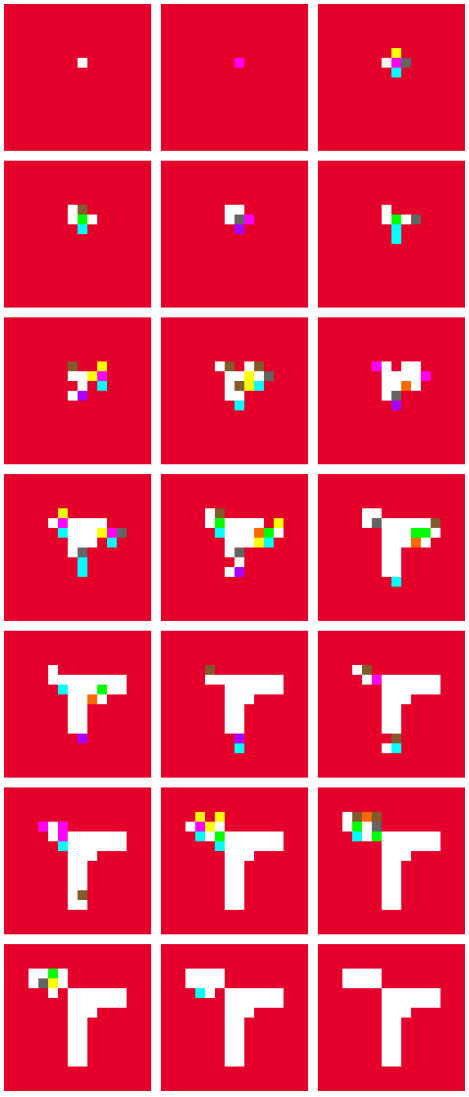
It is difficult to provide statistical data from these experiments since only very few successful results have been obtained so far and there has been a research in progress regarding the optimization of evolutionary techniques used for the design of complex cellular automata. Therefore, only some selected results are presented in this section.

The first experiment — the development of the BUT logo — dealt with the CA working with 10 cell states. There are 10^5 various combinations in the 5-cell neighborhood which implies 10^{100000} possible transition functions. A single state-1 cell (a seed) was used as an initial state of the CA. The evolution managed to find a transition function (in the form of a sequence of CMRs that was converted to the table-based representation for the presentation purposes) that successfully develops the initial seed into the given pattern as shown in Figure 9a. Although the pattern stability was not required, the pattern no longer changes by further CA development. A sample of the complete sequence of states leading to the emergence of this pattern (the BUT logo) is shown in Figure 10 together with the appropriate transition function. The CA uses 90 transition rules and needs 20 steps to finish the pattern.

The second experiment — the development of the author's surname — delat with the CA working with 12 cell states. There are 12^5 various combinations in the 5-cell neighborhood which implies 10^{248832} possible transition functions. In fact, this pattern requires to develop several (separate) structures (letters) of non-zero cells which form the complete label. A sample of the CA development together with the transition function is shown in Figure 11. This is the only result obtained so far which is able to fully develop this pattern (from a triples of separate cells in states 1, 2 and 3 as an initial CA state – see the top-left corner of Fig. 11a). Similarly to the previous experiment, this pattern is also stable during further CA development which is especially interesting due to its increased complexity. The CA needs 32 steps to develop the target label from the initial state and the transition function consists of 161 rules (see Figure 11b).

5.1 Discussion

Although the evolution of multi-state 2D CA is much more difficult than the 1D CA, the experiments provided some successful results with various target patterns. The results presented in this section probably represent the first case when complex 2D CA with at least 10 cell states were automatically designed using an evolutionary algorithm in a task of the non-trivial exact pattern development. In addition to the CA shown in Fig. 10 and 11, the evolution succeeded in searching other patterns as well (e.g. French flag, Czech flag, moving labels, replicating objects or multi-state gliders). This indicates that the proposed design method may be applicable in a wider area of cellular automata. A limitation for a higher success rate of the evolutionary experiments probably lies in the

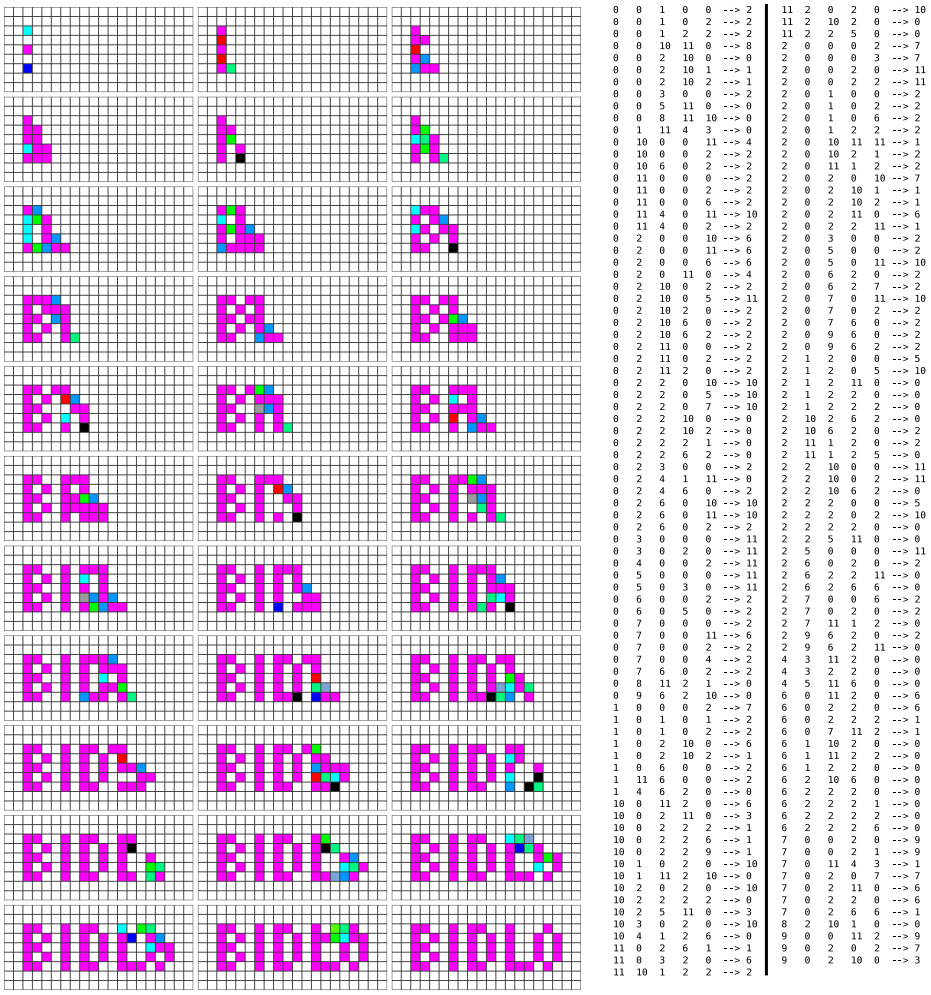


(a)

0	0	0	0	8	-->	3	1	1	3	9	0	-->	5
0	0	0	0	9	-->	3	1	1	4	0	1	-->	1
0	0	0	3	1	-->	1	1	1	4	3	1	-->	1
0	0	0	3	3	-->	1	1	1	5	1	0	-->	1
0	0	0	4	0	-->	1	1	1	5	2	3	-->	2
0	0	0	6	0	-->	1	1	1	7	0	0	-->	1
0	0	0	9	0	-->	1	1	1	8	0	1	-->	1
0	0	1	0	0	-->	9	1	1	8	1	0	-->	1
0	0	1	0	1	-->	4	1	1	8	3	0	-->	1
0	0	1	4	0	-->	9	1	1	8	9	7	-->	2
0	0	3	0	1	-->	4	1	2	2	1	1	-->	1
0	0	3	0	9	-->	4	1	3	6	0	0	-->	1
0	1	0	1	1	-->	1	1	5	1	0	0	-->	0
0	1	0	3	2	-->	1	1	6	1	0	0	-->	0
0	1	1	0	8	-->	0	2	0	6	0	0	-->	7
0	1	1	1	0	-->	9	2	0	6	0	6	-->	1
0	1	2	1	3	-->	1	2	0	6	1	0	-->	0
0	1	2	1	6	-->	1	2	1	5	1	0	-->	1
0	1	4	0	1	-->	1	2	3	6	0	0	-->	1
0	1	4	0	2	-->	1	2	5	1	0	0	-->	0
0	1	4	5	2	-->	1	2	8	3	1	1	-->	1
0	1	8	0	0	-->	9	3	1	0	6	0	-->	3
0	1	9	0	9	-->	1	3	1	9	1	6	-->	2
0	2	1	0	0	-->	9	3	1	9	3	6	-->	2
0	2	1	8	0	-->	3	3	1	9	8	6	-->	2
0	3	0	3	3	-->	5	3	3	1	0	2	-->	8
0	3	9	8	6	-->	2	3	3	9	0	6	-->	1
0	4	0	0	9	-->	9	3	4	3	6	0	-->	5
0	4	5	4	1	-->	2	4	1	1	0	6	-->	8
0	5	4	0	8	-->	1	4	1	2	1	6	-->	8
0	8	9	0	0	-->	1	4	1	6	0	0	-->	1
0	9	0	0	0	-->	8	4	1	8	0	2	-->	1
0	9	1	9	1	-->	3	5	1	3	6	0	-->	5
0	9	3	1	1	-->	1	5	2	1	8	1	-->	3
0	9	8	0	0	-->	1	6	0	6	0	0	-->	7
1	0	0	1	1	-->	1	7	0	0	0	0	-->	6
1	0	0	4	1	-->	1	8	0	0	0	0	-->	6
1	0	1	0	7	-->	4	8	0	0	1	0	-->	6
1	0	6	0	0	-->	7	8	0	6	0	6	-->	1
1	0	6	1	0	-->	0	8	0	7	0	0	-->	6
1	0	7	0	6	-->	4	8	1	2	1	0	-->	0
1	1	1	5	0	-->	5	9	0	0	0	0	-->	6
1	1	1	9	1	-->	3	9	0	0	1	0	-->	6
1	1	2	1	1	-->	1	9	1	0	0	0	-->	6
1	1	3	1	3	-->	1	9	1	9	1	6	-->	2

(b)

Fig. 10. Example of an evolved CA for the development of the BUT logo. The initial state consists of a single cell state 1, the other cells possess state 0. (a) The sequence of CA states producing the final pattern (ordered from left to right and top to bottom). (b) The appropriate transition rules for this CA.



(a)

```
0  0  1  0  0 --> 2
0  0  1  0  2 --> 2
0  0  1  2  2 --> 2
0  0 10 11  0 --> 8
0  0  2 10  0 --> 0
0  0  2 10  1 --> 1
0  0  2 10  2 --> 1
0  0  3  0  0 --> 2
0  0  5 11  0 --> 0
0  0  8 11 10 --> 0
0  1 11  4  3 --> 0
0 10  0  0 11 --> 4
0 10  0  0  2 --> 2
0 10  6  0  2 --> 2
0 11  0  0  0 --> 2
0 11  0  0  2 --> 2
0 11  0  0  6 --> 2
0 11  4  0 11 --> 10
0 11  4  0  2 --> 2
0  2  0  0 10 --> 6
0  2  0  0 11 --> 6
0  2  0  0  6 --> 6
0  2  0 11  0 --> 4
0  2 10  0  2 --> 2
0  2 10  0  5 --> 11
0  2 10  2  0 --> 2
0  2 10  6  0 --> 2
0  2 10  6  2 --> 2
0  2 11  0  0 --> 2
0  2 11  0  2 --> 2
0  2 11  2  0 --> 2
0  2  2  0 10 --> 10
0  2  2  0  5 --> 10
0  2  2  0  7 --> 10
0  2  2 10  0 --> 0
0  2  2 10  2 --> 0
0  2  2  2  1 --> 0
0  2  2  6  2 --> 0
0  2  3  0  0 --> 2
0  2  4  1 11 --> 0
0  2  4  6  0 --> 2
0  2  6  0 10 --> 10
0  2  6  0 11 --> 10
0  2  6  0  2 --> 2
0  3  0  0  0 --> 11
0  4  0  0  2 --> 11
0  5  0  0  0 --> 11
0  5  0  3  0 --> 11
0  6  0  0  2 --> 2
0  6  0  5  0 --> 2
0  7  0  0  0 --> 2
0  7  0  0 11 --> 6
0  7  0  0  2 --> 2
0  7  0  0  4 --> 2
0  7  6  0  2 --> 2
0  8 11  2  1 --> 0
0  9  6  2 10 --> 0
1  0  0  0  2 --> 7
1  0  1  0  1 --> 2
1  0  1  0  2 --> 2
1  0  2 10  0 --> 6
1  0  2 10  2 --> 1
1  0  6  0  0 --> 2
1 11  6  0  0 --> 2
1  4  6  2  0 --> 0
10  0 11  2  0 --> 6
10  0  2 11  0 --> 3
10  0  2  2  2 --> 1
10  0  2  2  6 --> 1
10  0  2  2  9 --> 1
10  1  0  2  0 --> 10
10  1 11  2 10 --> 0
10  2  0  2  0 --> 10
10  2  2  2  2 --> 0
10  2  5 11  0 --> 3
10  3  0  2  0 --> 10
10  4  1  2  6 --> 0
11  0  2  6  1 --> 1
11  0  3  2  0 --> 6
11 10  1  2  2 --> 2
```

```
11  2  0  2  0 --> 10
11  2 10  2  0 --> 0
11  2  2  5  0 --> 0
 2  0  0  0  2 --> 7
 2  0  0  0  3 --> 7
 2  0  0  2  0 --> 11
 2  0  0  2  2 --> 11
 2  0  1  0  0 --> 2
 2  0  1  0  2 --> 2
 2  0  1  0  6 --> 2
 2  0  1  2  2 --> 2
 2  0 10 11 11 --> 1
 2  0 10  2  1 --> 2
 2  0 11  1  2 --> 2
 2  0  2  0 10 --> 7
 2  0  2  2 10 --> 1
 2  0  2 10  2 --> 1
 2  0  2 11  0 --> 6
 2  0  2  2 11 --> 1
 2  0  3  0  0 --> 2
 2  0  5  0  0 --> 2
 2  0  5  0 11 --> 10
 2  0  6  2  0 --> 2
 2  0  6  2  7 --> 2
 2  0  7  0 11 --> 10
 2  0  7  0  2 --> 2
 2  0  7  6  0 --> 2
 2  0  9  6  0 --> 2
 2  0  9  6  2 --> 2
 2  1  2  0  0 --> 5
 2  1  2  0  5 --> 10
 2  1  2 11  0 --> 0
 2  1  2  2  0 --> 0
 2  1  2  2  2 --> 0
 2 10  2  6  2 --> 0
 2 10  6  2  0 --> 2
 2 11  1  2  0 --> 2
 2 11  1  2  5 --> 0
 2  2 10  0  0 --> 11
 2  2 10  0  2 --> 11
 2  2 10  6  2 --> 0
 2  2  2  0  0 --> 5
 2  2  2  0  2 --> 10
 2  2  2  2  0 --> 0
 2  2  5 11  0 --> 0
 2  5  0  0  0 --> 11
 2  6  0  2  0 --> 2
 2  6  2  2 11 --> 0
 2  6  2  6  6 --> 0
 2  7  0  0  6 --> 2
 2  7  0  2  0 --> 2
 2  7 11  1  2 --> 0
 2  9  6  2  0 --> 2
 2  9  6  2 11 --> 0
 4  3 11  2  0 --> 0
 4  3  2  2  0 --> 0
 4  5 11  6  0 --> 0
 6  0 11  2  0 --> 6
 6  0  2  2  0 --> 6
 6  0  2  2  2 --> 1
 6  0  7 11  2 --> 1
 6  1 10  2  0 --> 0
 6  1 11  2  2 --> 0
 6  1  2  2  0 --> 0
 6  2 10  6  0 --> 0
 6  2  2  2  0 --> 0
 6  2  2  2  1 --> 0
 6  2  2  2  2 --> 0
 6  2  2  2  6 --> 0
 7  0  0  2  0 --> 9
 7  0  0  2  1 --> 9
 7  0 11  4  3 --> 1
 7  0  2  0  7 --> 7
 7  0  2 11  0 --> 6
 7  0  2  2  0 --> 6
 7  0  2  6  6 --> 1
 8  2 10  1  0 --> 0
 9  0  0 11  2 --> 9
 9  0  2  0  2 --> 7
 9  0  2 10  0 --> 3
```

(b)

Fig. 11. Example of an evolved CA for the development of the author's surname. The initial state consists of three vertically alligned cells in states 1, 2 and 3 respectively with a single state-0 cells between them. (a) The sequence of CA states producing the final "BIDLO" pattern (ordered from left to right and top to bottom). (b) The appropriate transition rules for this CA.

requirement of an exact pattern development. Our initial experiments suggest that promising areas for the CA applications may be those where approximate results are acceptable or the CA states allow us to tolerate some variations during the development. For example, the image processing, traffic prediction or design of approximative algorithms represent possible topics. Therefore, the future research will include modeling, simulation and optimization of such kinds of systems in cellular automata.

Even though the representation of the transition functions by means of conditional rules has proven a good applicability on various tasks, the optimization of the evolutionary algorithm used to search for the rules is probably still possible. This could not only improve the success rate of the evolutionary experiments but also reduce the computational effort and allow applying the concept of uniform computing platforms in real-world applications (e.g. with acceleration of the computations using modern reconfigurable technology).

6 Conclusions

In this study we have presented some advanced topics related to the evolution of complex multi-state cellular automata. In particular, an analysis of CA for the generic square calculations has been proposed in the first case study. The results showed that some various algorithms to perform this task exist in CA, which differ both in the complexity of resulting transition functions and the efficiency of the computation (i.e. the number of steps of the CA needed to produce the result). Moreover, our best results presented herein have overcome the known solution (Wolfram's squaring CA), providing a reduction of the number of steps by approximately 50%.

The second case study has dealt with the non-trivial pattern development problem in two-dimensional CA. Several results have been presented that provide an exact and stable pattern developed from a simple initial CA state. Cellular automata working with 10 and 12 cell states have been treated, which induce search spaces of enormous sizes. Despite low success rates of the evolution, the results obtained have shown that the automatic design of such CA is possible even though our ongoing experiments indicate that the evolutionary algorithm still provides a space for further optimization.

In general, the proposed results probably represent the first case of the automatic design of exact behaviour in CA with more than 10 cell states. We believe that these pieces of knowledge will allow us to further improve our design method and to apply cellular automata for modeling, simulation and optimization of real-world problems.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II), project IT4Innovations excellence in science – LQ1602, and from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center – LM2015070".

References

1. Basanta, D., Bentley, P., Miodownik, M., Holm, E.: Evolving cellular automata to grow microstructures. In: Genetic Programming, Lecture Notes in Computer Science, vol. 2610, pp. 1–10. Springer Berlin Heidelberg (2003)
2. Berlekamp, E.R., Conway, J.H., Guy, R.K.: Winning Ways for Your Mathematical Plays, 2nd Ed., Volume 4. A K Peters/CRC Press (2004)
3. Bidlo, M.: Investigation of replicating tiles in cellular automata designed by evolution using conditionally matching rules. In: 2015 IEEE International Conference on Evolvable Systems (ICES). pp. 1506–1513. Proceedings of the 2015 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE Computational Intelligence Society (2015)
4. Bidlo, M.: Evolution of generic square calculations in cellular automata. In: Proceedings of the 8th International Joint Conference on Computational Intelligence - Volume 3: ECTA. pp. 94–102. SciTePress - Science and Technology Publications (2016)
5. Bidlo, M.: On routine evolution of complex cellular automata. IEEE Transactions on Evolutionary Computation 20(5), 742–754 (2016)
6. Bidlo, M., Vasicek, Z.: Evolution of cellular automata with conditionally matching rules. In: 2013 IEEE Congress on Evolutionary Computation (CEC 2013). pp. 1178–1185. IEEE Computer Society (2013)
7. Codd, E.F.: Cellular Automata. Academic Press, New York (1968)
8. Durand, B., Rka, Z.: The game of life: Universality revisited. In: Mathematics and Its Applications, Volume 460 Cellular Automata. pp. 51–74. Springer Netherlands (1999)
9. Elmenreich, W., Fehérvári, I.: Evolving self-organizing cellular automata based on neural network genotypes. In: Proceedings of the 5th International Conference on Self-organizing Systems. pp. 16–25. Springer (2011)
10. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
11. Ilachinski, A.: Cellular Automata: A Discrete Universe. World Scientific (2001)
12. Jean-Yves Perrier, Moshe Sipper, J.Z.: Toward a viable, self-reproducing universal computer. Physica D 97(4), 335–352 (1996)
13. Lindgren, K., Nordahl, M.G.: Universal computation in simple one-dimensional cellular automata. Complex Systems 4(3), 299–318 (1990)
14. Mardiris, V., Sirakoulis, G., Karafyllidis, I.: Automated design architecture for 1-d cellular automata using quantum cellular automata. Computers, IEEE Transactions on 64(9), 2476–2489 (2015)
15. von Neumann, J.: The Theory of Self-Reproducing Automata. A. W. Burks (ed.), University of Illinois Press (1966)

16. Ninagawa, S.: Solving the parity problem with rule 60 in array size of the power of two. Journal of Cellular Automata 8(3–4), 189–203 (2013)
17. Rendell, P.: A universal turing machine in conway's game of life. In: 2011 International Conference on High Performance Computing and Simulation (HPCS). pp. 764–772 (2011)
18. Rendell, P.: A fully universal turing machine in Conway's game of life. Journal of Cellular Automata 9(1–2), 19–358 (2013)
19. Sahoo, S., Choudhury, P.P., Pal, A., Nayak, B.K.: Solutions on 1-d and 2-d density classification problem using programmable cellular automata. Journal of Cellular Automata 9(1), 59–88 (2014)
20. Sahu, S., Oono, H., Ghosh, S., Bandyopadhyay, A., Fujita, D., Peper, F., Isokawa, T., Pati, R.: Molecular implementations of cellular automata. In: Cellular Automata for Research and Industry. pp. 650–659. Lecture Notes in Computer Science, Vol. 6350, Springer (2010)
21. Sipper, M.: Quasi-uniform computation-universal cellular cutomata. In: Advances in Artificial Life, ECAL 1995, Lecture Notes in Computer Science, Vol. 929. pp. 544–554. Springer Berlin Heidelberg (1995)
22. Sridharan, K., Pudi, V.: Design of Arithmetic Circuits in Quantum Dot Cellular Automata Nanotechnology. Springer International Publishing Switzerland (2015)
23. Stefano, G.D., Navarra, A.: Scintillae: How to approach computing systems by means of cellular automata. In: Cellular Automata for Research and Industry. pp. 534–543. Lecture Notes in Computer Science, Vol. 7495, Springer (2012)
24. Suzudo, T.: Searching for pattern-forming asynchronous cellular automata – an evolutionary approach. In: Cellular Automata, Lecture Notes in Computer Science, vol. 3305, pp. 151–160. Springer Berlin Heidelberg (2004)
25. Tempesti, G.: A new self-reproducing cellular automaton capable of construction and computation. In: Advances in Artificial Life, Proc. 3rd European Conference on Artificial Life. pp. 555–563. Lecture Notes in Artificial Intelligence, Vol. 929, Springer (1995)
26. Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign IL (2002)
27. Yuns, J.B.: Achieving universal computations on one-dimensional cellular automata. In: Cellular Automata for Research and Industry. pp. 660–669. Lecture Notes in Computer Science Volume 6350, Springer (2010)