Journal of Computing and Information Technology - CIT 19, 2011, 2, 105–112
doi:10.2498/cit.1001419

Evolutionary Synthesis of Cellular Automata

Jernej Zupanc[1] and Bogdan Filipič[2]

[1] Faculty of Computer and Information Science, University of Ljubljana, Slovenia
[2] Department of Intelligent Systems, Jožef Stefan Institute, Ljubljana, Slovenia

Synthesis of cellular automata is an important area of modeling and describing complex systems. Large amounts of combinations and candidate solutions render the usage of deterministic approaches impractical and thus nondeterministic optimization methods have to be employed. Two of the typical evolutionary approaches to synthesizing cellular automata are the evolution of a single automaton and a genetic algorithm that evolves a population of automata. The first approach, with addition of some heuristics, is known as the cellular programming algorithm. In this paper we address the second approach and develop a genetic algorithm that evolves a population of cellular automata. We test both approaches on the density classification task, which is one of the most widely studied computational problems in the context of evolving cellular automata. Comparison of the synthesized cellular automata demonstrates unexpected similarity of the evolved rules and comparable classification accuracy performance of both approaches.

Keywords: cellular automata, cellular programming algorithm, density classification task, evolutionary computing, genetic algorithm

1. Introduction

Cellular automata are a framework for modeling complex systems. Even though they were introduced in 1940s with primarily experimental motives [1, 2], their modeling ability has since been applied in various research areas and real life problems, for instance: biological modeling [3], cryptography [4], traffic modeling [5], etc. The general approach to modeling with cellular automata is to use an optimization method for synthesizing its elements: the neighborhood and the rule table. Even in optimization of one element of a cellular automaton, the rule table, the space of candidate solutions is still immense. Therefore, a genetic algorithm and its derivatives are commonly employed to find a cellular automaton that best solves a given problem. One such derivative is the cellular programming algorithm (CPA) which begins with a random cellular automaton and synthesizes its rule table through multiple steps of evolution [6, 7]. This algorithm is known to perform well on tasks such as the density classification task (DCT) [8], the synchronization task [9], etc. In this paper, we consider a more general genetic algorithm (GA) to find the best cellular automaton by evolving a population of cellular automata. Both the proposed GA and the CPA are applied to the DCT. The main objective of this study is to compare the rules obtained with both approaches. The rules evolved with the CPA tend to be quasi-uniform with only a small subset of all possible single cell rules being present. As this has a positive effect on solving the DCT, we were interested in finding out whether such rules also dominate when cellular automata are evolved with the GA.

The remainder of this paper is organized as follows. First, an overview of the theoretical background on cellular automata is provided, followed by a description of the DCT. Next, we describe the CPA and the GA. Then we present numerical experiments and analyze the results. The paper concludes with a summary of the work and directions for further research.

2. Cellular Automata

Cellular automata were introduced by Ulam and Von Neumann in the 1940s as a formal framework to study behavior of complex and com-

prehensive systems [1, 2]. They are dynamic systems, discrete in time and space. Here, discrete means that space, time, and the properties of an automaton can only have a finite number of states. Although cellular automata can theoretically exist in n-dimensions, most commonly one-, two- and three-dimensional automata are used. A cellular automaton is a lattice of cells where each cell is a binary memory element. Moreover, a cellular automaton consists of the following elements: a rule table, information about each cell's neighborhood, and the state of the automaton. The rule of each cell defines what its state will be in the next time step, depending on the state of its neighborhood. In this context, different neighborhood definitions are possible, although in most cases cells near the observed cell in the lattice are chosen to be its neighbors and (together with the cell itself) represent the cell's neighborhood. The rules of a cellular automaton are presented in a rule table. In this respect, the behavior of all cells of the automaton is described. When all cells of a cellular automaton use the same rule, and therefore behave in the same way, the automaton and its rules are called uniform. Since in each step the states of the automaton cells are calculated from the rules and the cells neighborhoods, the states before the first step have to be defined differently. These states are called the initial configuration and typically depend on the problem being solved. When talking about synthesizing cellular automata, one has in mind finding the rules (or even neighborhoods) that best solve this problem. This is usually done with an optimization algorithm for searching the space of all possible rules. After the synthesis, the cellular automaton is run for a predefined number of steps and the states dynamics are observed. If required for the problem we are solving, the final states after the cellular automaton run are also observed. In this respect, the DCT is especially interesting.

As this paper focuses on approaches based on genetic algorithms, we frequently use the term *evolving an individual* or *evolving a cellular automaton*. Nevertheless, in this context the only element of the cellular automaton being evolved are its cell rules – the rule table.

3. Density Classification Task

The DCT is a computational problem proposed in 1978 by Gács, Kurdyumov, and Levin [8] and is also referred to as the majority task. To solve the DCT, a cellular automaton is required to classify a finite binary sequence based on the prevailing number of zeros or ones. Standard formulation of the DCT states that a binary, one-dimensional cellular automaton has to converge to a final configuration of all cells in state 1 when the initial configuration contains more ones than zeros, and to a configuration of all cells in state 0 when the initial configuration contains more zeros than ones. The answer to the majority problem is stored in the states of the cellular automaton after the run and is considered to be

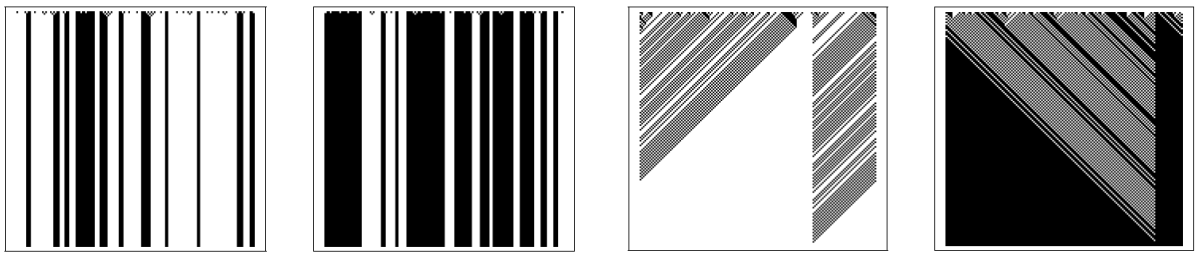

Figure 1. Four examples of solving the DCT with a one-dimensional cellular automaton consisting of 149 cells. Each graph represents the history of steps leading to the final solution. Solving proceeds vertically top-down, each line representing one step. The first line represents the initial configuration and the last line the solution computed with the cellular automaton. The images show solving the DCT: (a) with the majority rule 232 (Wolfram notation [6], Table 2) and the initial density of ones 0.4, (b) with the majority rule 232 and the initial density of ones 0.6, (c) with a heterogeneous rule table and an initial density of ones 0.4, (d) with a heterogeneous rule table and an initial density of ones 0.6.

the result of classification [2]. Solving this task with a computer that has centralized processing is trivial, however, let us once again draw attention to the fact that the next state of each cell in the cellular automaton is decided only upon the states of its neighborhood. Consequently, the processing is distributed, and solution cannot be obtained by simply counting the number of ones and ordering the cells of the cellular automaton into the correct state. The cells processing only local information have to solve the task as a whole and this makes solving the DCT with cellular automata challenging. Four examples of solving the DCT with cellular automata can be seen in Figure 1.

4. Cellular Programming Algorithm

An established algorithm for synthesizing cellular automata is the CPA, presented in Figure 2. It has been successfully applied to solving problems such as synchronization, square filling and the majority task [6, 7]. Although its core is the GA, there are a few specifics worth mentioning. Instead of evolving a population of candidate solutions, the CPA focuses on a single individual and evolves its rule table. This is not done by applying the crossover operator to the rule tables of a population of cellular automata. Instead, every single cell's fitness is compared to the fitness of its neighboring cells. Rules of the cells with low fitness are replaced with the rules obtained from better performing neighboring cells. The cell rules with the highest fitness among the neighbors are kept and in some cases replace those with lower fitness (Figure 3). As a result, rules that are more likely to lead to misclassification are eliminated and replaced with better ones from the neighborhood.

```
FOR each cell_i in CA DO
 Initialize cell_i rule table
 cell_i fitness f_i = 0
END FOR
c = 0
WHILE not done DO
 Generate random initial
configurations IC
 Run CA on all IC for M steps
 FOR each cell_i DO
   IF cell_i correctly classifies
THEN
     f_i = f_i + 1
   END IF
 END FOR
 Calculate fitness of CA from cell
fitness
 c = c + 1
 IF c mod C = 0 THEN
   FOR each cell_i DO
     Calculate nf_i(c)
     IF nf_i(c)=0 THEN
       No change
     ELSIF nf_i(c)=1 THEN
       Exchange with the better
       Mutation
     ELSIF nf_i(c)=2 THEN
       Exchange with crossover ...
       merging of better neighbors
       Mutation
     END IF
     f_i = 0
   END FOR
 END IF
END WHILE

f_i:  classification accuracy of cell_i
c:  counter of configurations
nf_i(c):  number of fitter neighbors
```

Figure 2. Pseudo code of the CPA.

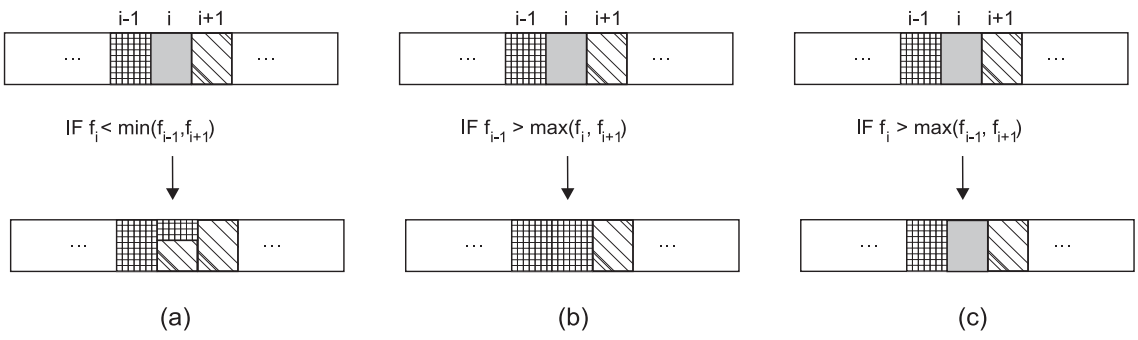

Figure 3. Examples of generating new individuals (rule tables) with the crossover operator in the CPA: (a) exchange with crossover of both better neighbors, (b) exchange with the single better neighbor, and (c) no change. The function f_i denotes the fitness for the cell i.

5. Genetic Algorithm

The other approach to the cellular automata synthesis is the GA, presented in Figure 4. This algorithm begins with a population of individuals and through applying evolutionary operators improves the average fitness of the population. Such approach leads to synthesizing exceptional individuals in the population.

GA is a population-based meta-heuristic optimization algorithm [10]. It belongs to a family of evolutionary algorithms and uses mechanisms inspired by biological evolution: reproduction, recombination, mutation, and selection, to find (near)optimal solutions to the problem considered. Candidate solutions to the optimization problem play the role of individuals in a population, and the cost function (also called the fitness function) determines the environment within which the solutions exist. Through repeated application of the genetic operators over the population of candidate solutions it is expected that average fitness of the population increases. After multiple generations, such repetition usually leads to finding at least suboptimal solutions to the problem as exceptional individuals are likely to appear in the population.

As both the CPA and the GA come from the family of evolutionary algorithms, the difference between the two can also be viewed from the perspective of the two major approaches to evolutionary computing: the Michigan [12] and the Pittsburgh [13] approach. The CPA can be associated with the Michigan approach since it evolves a single individual. This approach provides an interesting property since although all rules compete to participate in the next generation, they also have to cooperate to establish a rule base that performs well with respect to the objective.

A drawback of the Michigan approach is that evolving a single individual and constant application of a priori knowledge can lead to less diverse solutions. The Pittsburgh approach, on the other hand, applies evolutionary operators to a population of individuals, which compete for involvement in the next generation. As a result, a greater diversity in the population pool is achieved, since there are no limitations to the structure of individual cellular automata. Our GA uses the Pittsburgh concept as we believe

```
g = 1
P(g) = population of CA's
FOR each CA in P(g)
 FOR each cell_i in CA
   Initialize cell_i rule table
   cell_i fitness f_i = 0
 END FOR
END FOR
WHILE not done DO
 Generate random initial...
configurations IC
 FOR each CA in P(g)
   Run CA on all IC for M steps
   FOR each cell_i in CA DO
     IF cell_i correctly class. THEN
       f_i = f_i + 1
     END IF
   END FOR
   Calculate fitness of CA...
   from cell fitness
 END FOR
 % Tournament selection
 Select P'(g) from P(g)
 % crossover of CA's
 FOR pairs of CA from P'(g)
   Crossover at random place...
   in CA rule table
 END FOR
 % mutation
 Randomly mutate bits ...
 in rule tables of P'(g)
 g = g+1
 P(g) = P'(g-1)
END WHILE

f_i:  classification accuracy of cell_i
g:  counter of generations
```

Figure 4. Pseudo code of the GA for evolving populations of cellular automata.

that the advantages of a diverse population pool should more than compensate the lack of not imposing any limitations to the structure of solutions.

6. Experiments and Results

6.1. Experimental setup

Let us first focus on the properties of the cellular automata used in the experiments. As listed in Table 1, the cellular automata are one-dimensional and consist of 19 cells. The cell neighborhood contains the cell i together with its left neighbor $i-1$ and right neighbor $i+1$. This way, a neighborhood of size 3 is formed. The outmost cells (number 1 and 19) are considered neighbors. A one-dimensional *line* cellular automaton can therefore be viewed as a *circle*, so every cell has its complete neighborhood.

Property	Value
Number of cells	19
Neighborhood size	3
Cell rule length	8
Cell states	Binary
Cell rules	Nonuniform
Number of dimensions	1

Table 1. Properties of the cellular automata used in the experiments.

Since the behavior of each cell is decided upon the states of its neighborhood cells and the states are binary, the size of every cell rule is 2^3 bits. A widely accepted notation for rules is the Wolfram notation, presented in Table 2, which uses the decimal system to describe binary presented rules [6].

Cell neighborhood	New value of the cell
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Table 2. The majority rule, also called rule 232 in Wolfram notation, for a one-dimensional cellular automaton with a neighborhood of 3. If we join the bits in the right column bottom-up into a word, it can be interpreted as $11101000_2 = 232_{10}$.

As no framework to force uniform rules among the cells is imposed in the GA approach, a cellular automaton can be synthesized so that every cell behaves according to its own rule. Consequently, the size of a single rule table as an individual in the GA is 152 bits, and the solution pool consists of 2^{152} possible individuals. Furthermore, to evaluate the quality of individuals in the GA, there is a need for a fitness function. To evaluate a cellular automaton, the latter has to solve the DCT for a number of random initial configurations, and its fitness is based on the classification accuracy. Duration of a single run is chosen to be $M = 21$ steps, which was experimentally proven to be enough for the cellular automaton to reach a fixed state. After each run, each cell of the cellular automaton is given a score of 1 if the cell is in the correct state and 0 otherwise. The fitness of a cell is accumulated over all different initial configurations, where 100% fitness after the last run means that a cell was in the correct state at the end of the DCT for all initial configurations. There is only a slight difference between the fitness functions of the GA and the CPA: while in the CPA a single cell's rule competes against its neighbors, individuals in the GA are entire rule tables – therefore their fitness is calculated as an average of all individual cells' fitness. Another fact to note is that all initial configurations are generated randomly so that all different densities of ones are equally probable. In this way, we avoid the difficulties which could occur if zeros and ones in the cells of all initial configurations were equally probable. In such a case, finding the optimal rules with any evolutionary approach is not feasible. The small bit-difference between configurations would often cause configurations with small majorities of zeros or ones to be wrongly classified and thus render the optimization algorithm useless.

During the synthesis of rules, each cellular automaton was tested on 500 random initial configurations. Both algorithms were run 100 times with the algorithm parameter settings distributed randomly in such a way that their combinations approximately covered the ranges where the algorithms are known to perform well. Results presented in this paper are obtained with the best cellular automata each of the algorithms synthesized. The algorithms performed best when their parameters were set as follows:

- number of generations $= 10^4$,
- number of fitness tests $= 500$,
- probability of mutation $= 0.01$,
- probability of crossover $(\mathrm{GA}) = 0.7$,
- probability of crossover $(\mathrm{CPA}) = 0.9$,
- number of crossover points $(\mathrm{GA}) = 4$,
- number of crossover points $(\mathrm{CPA}) = 1$,
- population size $(\mathrm{GA}) = 200$,
- tournament size $(\mathrm{GA}) = 50$.

Further, let us take a look at both algorithms and the specifics of the experiments. For the GA (Figure 4), a population of 200 rule tables is randomly chosen so that zeros and ones are equally represented and no specific patterns are imposed. This is done only for the initial population, as later in the evolution new individuals evolve from their parents. Each individual is administered to the fitness function for evaluation of the quality of solving the DCT. After fitness of all the individuals is examined, tournament selection is used to select the individuals for the mating pool of the subsequent generation [11]. Every parent for the next generation is chosen as the best individual (highest fitness) among randomly chosen representatives in the tournament. The size of the tournament is selected experimentally and set at 50. The crossover operator is then applied to the mating pool where pairs of individuals are split apart at four random points and the parts are recombined into a pair of new individuals. These are exposed to mutation, which is executed randomly, bitwise. The individuals generated through these steps represent a new population and are subjects to fitness evaluation.

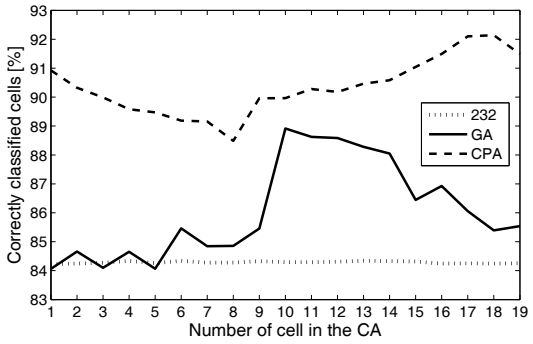

Figure 5. Fitness of the uniform CA 232 and the CA evolved with the GA and the CPA on 10^6 initial configurations.

6.2. Results and comments

Here we present the best cellular automata synthesized by both algorithms and their properties, and comment on the results. At this point, it should be explained what is meant by the fitness of a population in both cases. When dealing with the CPA, the fitness of a population is the fitness of the cellular automaton being evolved, as there is only one, while in the context of the GA it is the maximum fitness that any individual cellular automaton in the observed population achieves.

It can be seen (Figure 7) that after about 200 generations, when the fitness of the population evolved with the GA reaches a plateau, the fitness achieved with the CPA keeps improving. It is to be noted that the horizontal axis is logarithmic and that the differences in the rate of change of fitness are not so obvious on the linear scale. At the end of the evolutionary synthesis, the best fitness achieved with the CPA is 91% classification accuracy, whereas with the GA it is 89%.

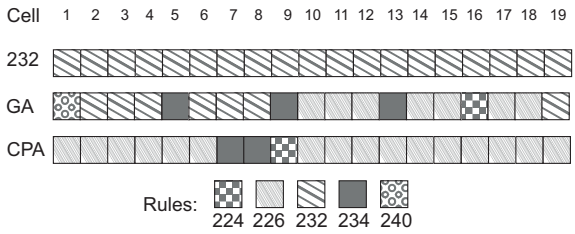

Figure 6. The uniform 232 rule table and the rule tables evolved with the two algorithms. Cell numbers correspond to successive numbers of the cell rules in the cellular automaton, and the rule numbers denote the cell rules in Wolfram notation.

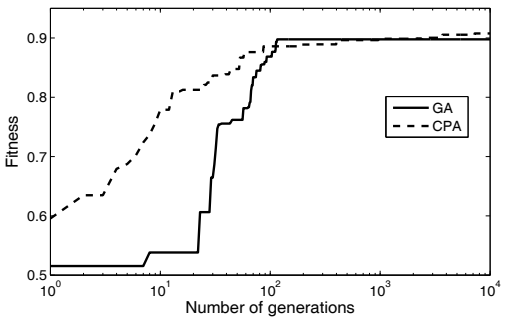

Figure 7. Best cellular automaton fitness during the evolution with both approaches.

The best cellular automata synthesized with both algorithms, together with the uniform 232 rule table, are presented in Figure 6. Both approaches find rule tables which can be described as "quasi-uniform rules of the cellular automata", meaning that the majority of cells use only a small subset of possible rules. Because of the nature of the CPA that propagates the best rules across the neighboring cells, quasi-uniformness of its resulting cellular automata is expected. On the other hand,

it is somewhat surprising to find such quasi-uniformness in the cellular automata evolved with the GA. In contrast to the CPA, the GA provides no framework for propagating the cell rules between the cells and neither the crossover operator implies such behavior.

The best cellular automaton each algorithm evolves (in 100 runs with different algorithm parameter settings) is tested over 10^6 random initial configurations which are equal for both algorithms. These runs on test initial configurations are performed to compare the performance of individual cell rules inside each cellular automaton rule table. The reason for comparing the efficiency of the cell rules was the unusually high occurrence of the rule 226 in the cellular automaton evolved with the CPA. From Figure 6 and Figure 5 it is evident which cell rules perform best. On the horizontal axis in Figure 5, the leftmost cell in the automaton is labeled as 1 and the rightmost cell is labeled as 19. In the same graph, it is evident that both synthesized cellular automata classify the tested initial configurations noticeably better than the uniform 232 cellular automaton. The classification accuracy analysis of the cells in the uniform 232 cellular automata shows that all cells have about the same, 84% classification accuracy. Their performance is similar to the cells with the same 232 rule in the cellular automata synthesized with the GA. As the latter is quasi-uniform, a noticeable increase in cell fitness can be seen between the cells 10 and 14 where the combination of rules 226 and 234 dominates. It is interesting that these two rules are dominantly present also in the cellular automaton synthesized with the CPA. The combination of rules 226, 234 and 224 is the best cellular automaton evolved in the experiments. Almost all of its cells exceed the classification accuracy of 90% and only one of the 234 cells falls below the 90% mark. Interestingly, both best cellular automata the evolutionary approaches synthesized were quasi-uniform.

7. Conclusion

Cellular automata have the ability to describe and model complex systems. Two approaches to the cellular automata synthesis tested in this study are the CPA and the GA. The experiments demonstrate that both algorithms are able to synthesize cellular automata capable of successfully performing the density classification task. When tested on the randomly generated initial configurations, the average fitness of both evolved cellular automata exceeded the fitness achieved with the best known uniform cellular automata with the rule 232. An interesting property of the best cellular automata evolved with the two algorithms is that both have quasi-uniform rules. This is somewhat expected for the CPA, but comes as a surprise with the GA. The resulting cellular automata rules consist of only five distinct rules which is a small amount, since the maximum number of distinct rules both algorithms could evolve is 38 (sum of the number of cells in both cellular automata). The most frequent rule in both resulting automata is the rule 226, which is not the rule that constitutes the best uniform cellular automata (232 from the literature). In addition, it was observed that the fitness curve of the GA reaches a plateau after a certain number of generations, while with the CPA it further improves.

Performance improvement of the GA could possibly be achieved by adaptable algorithm parameters that would allow the algorithm to smoothly converge to an even better solution. Moreover, decreasing the probability of crossover and mutation and decreasing the number of crossover points in later generations could also result in better convergence. Another interesting issue for future work would be merging both approaches: first synthesizing a good solution with the CPA and then passing it to the GA for further improvement. In this way, the advantage of individual rule propagation between cells in the CPA would be merged with the diverse population property of the GA for fine tuning.

References

[1] J. VON NEUMANN, *Theory of Self-reproducing Automata*. University of Illinois Press, Illinois, 1966.

[2] M. S. CAPCARERRE, M. SIPPER, M. T, Two-state, r=1 cellular automaton that classifies density. *Physical Review Letters*, 77:24 (1996), pp. 4969–4971.

[3] G. B. ERMENTROUT, L. EDELSTEIN-KESHET, Cellular automata approaches to biological modeling. *Journal of Theoretical Biology*, 160 (1993), pp. 97–133.

[4] S. NANDI, B. K. KAR, P. CHAUDHURI, Theory and applications of cellular automata in cryptography. *IEEE Transactions on Computers*, 43:12 (1994), pp. 1346–1357.

[5] G. GUOQING, H. POMING, W. BINGHONG, D. SHIQIANG, Two-dimensional cellular automaton traffic model with randomly switching traffic lights. *Applied Mathematics and Mechanics*, 19:9 (1998), pp. 807–813.

[6] M. SIPPER, *Evolution of Parallel Cellular Machines – The Cellular Programming Approach*. Springer-Verlag, Heidelberg, 1997.

[7] M. SIPPER, *Evolution of Parallel Cellular Machines*. Moshe Sipper, 2004.

[8] P. GACS, G. L. KURDYUMOV, L. A. LEVIN, One dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii*, 12 (1978), pp. 92–98.

[9] R. DAS, J. P. CRUTCHFIELD, M. MITCHELL, J. E. HANSON, Evolving globally synchronized cellular automata. In *Proceedings of The Sixth International Conference on Genetic Algorithms*, (1995), pp. 336–343.

[10] J. H. HOLLAND, *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Michigan, 1975.

[11] D. E. GOLDBERG, A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4 (1990), pp. 445–460.

[12] A. BONARINI, Evolutionary learning of fuzzy rules: Competition and cooperation. In: W. Pedrycz, Editor, *Fuzzy Modeling: Paradigms and Practice*, Kluwer Academic, Norwell, Mass. (1996), pp. 265–284.

[13] S. F. SMITH, A learning system based on genetic adaptive algorithms. PhD. dissertation, Department of Computer Science, University of Pittsburgh, 1980.

Received: January, 2009
Accepted: April, 2011

Contact addresses:

Jernej Zupanc
Faculty of Computer and Information Science
University of Ljubljana
Tržaška cesta 25
SI-1000 Ljubljana
Slovenia
e-mail: jernej.zupanc@fri.uni-lj.si

Bogdan Filipič
Department of Intelligent Systems
Jožef Stefan Institute
Ljubljana, Slovenia
e-mail: jernej.zupanc@fri.uni-lj.si

JERNEJ ZUPANC is a junior researcher at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. He received his BSc in computer and information science in 2006. His research interests include evolutionary computation, neural networks and semi-supervised learning.

BOGDAN FILIPIČ received his PhD in computer science from the University of Ljubljana, Slovenia, in 1993. He is a senior researcher at the Department of Intelligent Systems of the Jožef Stefan Institute, Ljubljana. His research interests include evolutionary computation, intelligent data analysis and knowledge-based systems.