Inverse Design of Cellular Automata by Genetic Algorithms: An Unconventional Programming Paradigm

Thomas Bäck[1,2], Ron Breukelaar[1,*], Lars Willmes[2]

[1] Universiteit Leiden, LIACS, P.O. Box 9512, 2300 RA Leiden, The Netherlands
{baeck,rbreukel}@liacs.nl
[2] NuTech Solutions GmbH, Martin Schmeißer Weg 15, 44227 Dortmund, Germany
{baeck,willmes}@nutechsolutions.de

Abstract. Evolving solutions rather than computing them certainly represents an unconventional programming approach. The general methodology of evolutionary computation has already been known in computer science since more than 40 years, but their utilization to program other algorithms is a more recent invention. In this paper, we outline the approach by giving an example where evolutionary algorithms serve to program cellular automata by designing rules for their iteration. Three different goals of the cellular automata designed by the evolutionary algorithm are outlined, and the evolutionary algorithm indeed discovers rules for the CA which solve these problems efficiently.

1 Evolutionary Algorithms

Evolutionary Computation is the term for a subfield of Natural Computing that has emerged already in the 1960s from the idea to use principles of natural evolution as a paradigm for solving search and optimization problem in high-dimensional combinatorial or continuous search spaces. The algorithms within this field are commonly called evolutionary algorithms, the most widely known instances being genetic algorithms [6, 4, 5], genetic programming [8, 9], evolution strategies [12–15], and evolutionary programming [3, 2]. A detailed introduction to all these algorithms can be found e.g. in the Handbook of Evolutionary Computation [1].

Evolutionary Computation today is a very active field involving fundamental research as well as a variety of applications in areas ranging from data analysis and machine learning to business processes, logistics and scheduling, technical engineering, and others. Across all these fields, evolutionary algorithms have convinced practitioners by the results obtained on hard problems that they are very powerful algorithms for such applications. The general working principle of all instances of evolutionary algorithms today is based on a program loop that

* Part of the research was funded by the Foundation for Fundamental Research on Matter (FOM), Utrecht, The Netherlands, *project: "An evolutionary approach to many-parameter physics"*.

involves simplified implementations of the operators mutation, recombination, selection, and fitness evaluation on a set of candidate solutions (often called a population of individuals) for a given problem. In this general setting, mutation corresponds to a modification of a single candidate solution, typically with a preference for small variations over large variations. Recombination corresponds to an exchange of components between two or more candidate solutions. Selection drives the evolutionary process towards populations of increasing average fitness by preferring better candidate solutions to proliferate with higher probability to the next generation than worse candidate solutions. By fitness evaluation, the calculation of a measure of goodness associated with candidate solutions is meant, i.e., the fitness function corresponds to the objective function of the optimization problem at hand.

This short paper does not intend to give a complete introduction to evolutionary algorithms, as there are many good introductory books on the topic available and evolutionary algorithms are, meanwhile, quite well known in the scientific community. Rather, we would like to briefly outline the general idea to use evolutionary algorithms to solve highly complex problems of parameterizing other algorithms, where the evolutionary algorithm is being used to find optimal parameters for another algorithm to perform its given task at hand as good as possible. One could also view this as an inverse design problem, i.e., a problem where the target design (behavior of the algorithm to be parameterized) is known, but the way to achieve this is unknown. The example we are choosing in this paper is the design of a rule for a 2 dimensional cellular automaton (CA) such that the cellular automaton solves a task at hand in an optimal way. We are dealing with 2 dimensional CA where the cells have just binary states, i.e., can have a value of one or zero. The behavior of such a CA is fully characterized by a rule which, for each possible pattern of bit values in the local neighborhood of a cell (von Neumann neighborhood: the cell plus its four vertical and horizontal direct nearest neighbors; Moore neighborhood: the cell plus its 8 nearest neighbors, also including the diagonal cells), defines the state of this cell in the next iteration of the CA evolution process. In the next section, we will explain the concept of a CA in some more detail. Section 5 reports experimental results of our approach with a 5 by 5 CA where the goal is to find rules which evolve from a standardized initial state of the CA to a target bit pattern, such that the rule rediscovers (i.e., inversely designs) this bit pattern. Finally, we give some conclusions from this work.

2 Cellular Automata

According to [16] Cellular Automata (CA) are mathematical idealizations of physical systems in which space and time are discrete, and physical quantities take on a finite set of discrete values. The simplest CA is one dimensional and looks a bit like an array of ones and zeros of width N, where the first position of the array is linked to the last position. In other words, defining a row of positions $C = \{a_1, a_2, ..., a_N\}$ where C is a CA of width N and a_N is adjacent to a_1.

The neighborhood s_n of a_n is defined as the local set of positions with a distance to a_n along the connected chain which is no more than a certain radius (r). This for instance means that $s_2 = \{a_{148}, a_{149}, a_1, a_2, a_3, a_4, a_5\}$ for $r = 3$ and $N = 149$. Please note that for one dimensional CA the size of the neighborhood is always equal to $2r + 1$.

The values in a CA can be altered all at the same time (synchronous) or at different times (asynchronous). Only synchronous CA are considered in this paper. In the synchronous approach at every timestep (t) every cell state in the CA is recalculated according to the states of the neighborhood using a certain transition rule $\Theta : \{0,1\}^{2r+1} \rightarrow \{0,1\}, s_i \rightarrow \Theta(s_i)$. This rule basically is a one-to-one mapping that defines an output value for every possible set of input values, the input values being the 'state' of a neighborhood. The state of a_n at time t is written as a_n^t, the state of s_n at time t as s_n^t and the state of the entire CA C at time t as C^t so that C^0 is the initial state and $\forall n = 1, \ldots, N \; a_n^{t+1} = \Theta(s_n^t)$. Given $C^t = \{a_1^t, ..., a_N^t\}$, C^{t+1} can be defined as $\{\Theta(s_1^t), ..., \Theta(s_N^t)\}$.

Because $a_n \in \{0,1\}$ the number of possible states of s_n equals 2^{2r+1}. Because all possible binary representations of m where $0 \leq m < 2^{2r+1}$ can be mapped to a unique state of the neighborhood, Θ can be written as a row of ones and zeros $R = \{b_1, b_2, ..., b_{2^{2r+1}}\}$ where b_m is the output value of the rule for the input state that maps to the binary representation of $m - 1$. A rule therefore has a length that equals 2^{2r+1} and so there are $2^{2^{2r+1}}$ possible rules for a binary one dimensional CA. This is a huge number of possible rules (if $r = 3$ this sums up to about $3,4 \times 10^{28}$) each with a different behavior.

One of the interesting things about these and other CA is that certain rules tend to exhibit organizational behavior, independently of the initial state of the CA. This behavior also demonstrates there is some form of communication going on in the CA over longer distances than the neighborhood allows directly. In [10] the authors examine if these simple CA are able to perform tasks that need positions in a CA to work together and use some form of communication. One problem where such a communication seems required in order to give a good answer is the Majority Problem (as described in section 4.1). A genetic algorithm is used to evolve rules for one dimensional CA that do a rather good job of solving the Majority Problem [10] and it is shown how these rules seem to send "particles" and communicate by using these particles [11]. These results imply that even very simple cells in one dimensional cellular automata can communicate and work together to form more complex and powerful behavior.

It is not unthinkable that the capabilities of these one dimensional CA are restricted by the number of directions in which information can "travel" through a CA and that using multiple dimensions might remove these restriction and therefore improve performance. Evolving these rules for the Majority Problem for two dimensional CA using a Moore neighborhood (explained in section 4) is reported in [7] showing that the GA did not clearly outperform random search.

The goal of the research is to find a generalization and report phenomena observed on a higher level, with the future goal to use this research for identification and calibration of higher-dimensional CA applications to real world

systems like parallel computing and modeling social and biological processes. The approach is described and results are reported on simple problems such as the Majority Problem, AND, XOR, extending into how it can be applied to pattern generation processes.

3 The Genetic Algorithm

As mentioned before, this research was inspired by earlier work [10, 11] in which transition rules for one dimensional CA were evolved to solve the Majority Problem (as defined in section 4.1). The GA is a fairly simple algorithm using binary representation of the rules, mutation by bit inversion, truncation selection, and single-point crossover. The algorithm determined the fitness by testing the evolved rules on 100 random initial states. Every iteration the best 20% of the rules (the 'elite' rules) were copied to the next generation and the other 80% of the rules were generated using single-point crossover with two randomly chosen 'elite' rules and then mutated by flipping exactly 2 bits in the rule.

To be able to compare two dimensional CA with one dimensional CA the GA used in section 4.1 is a copy of the the GA used in [10, 11]. The GA's in section 4.2 and 5 on the other hand are modified to fit the different problem demands, as will be explained in these sections.

4 Experimental results for two dimensional CA

The two dimensional CA in this document are similar to the one dimensional CA discussed so far. Instead of a row of positions, C now consists of a grid of positions. The values are still only binary (0 or 1) and there still is only one transition rule for all the cells. The number of cells is still finite and therefore CA discussed here have a width, a height and borders.

The big difference between one dimensional and two dimensional CA is the rule definition. The neighborhood of these rules is two dimensional, because there are not only neighbors left and right of a cell, but also up and down. That means that if $r = 1$, s_n would consist of 5 positions, being the four directly adjacent plus a_n. This neighborhood is often called "the von Neumann neighborhood" after its inventor. The other well known neighborhood expands the Neumann neighborhood with the four positions diagonally adjacent to a_n and is called "the Moore neighborhood" also after its inventor.

Rules are defined with the same rows of bits (R) as defined in the one dimensional case. For a von Neumann neighborhood a rule can be defined with $2^5 = 32$ bits and a rule for a Moore neighborhood needs $2^9 = 512$ bits. This makes the Moore rule more powerful, for it has a bigger search space. Yet, a bigger search space also implies a longer search time and finding anything usefull might be a lot more difficult. In [7] the authors discourage the use of the Moore neighborhood, yet in section 4.2 and section 5 results clearly show successes using the Moore neighborhood, regardless of the larger search space.

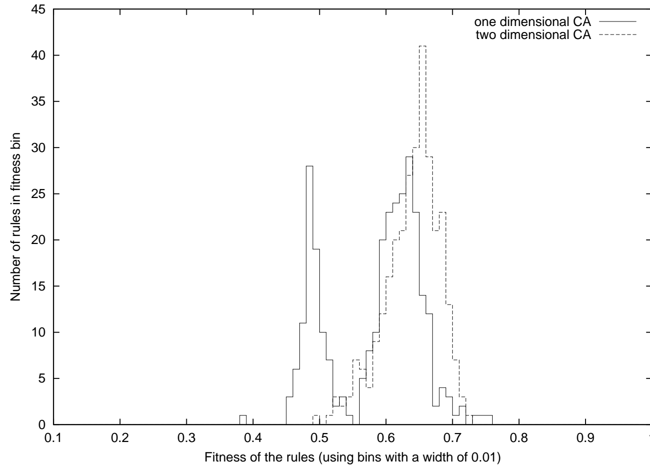

Fig. 1. This figure displays the number of rules that have a certain fitness value in the two dimensional experiment and compares this to the one dimensional experiment. The fitness bins are 0.01 in width and for both algorithm $F_{169,10^3}$ is calculated for 300 rules.

In a one dimensional CA the leftmost cell is connected to the rightmost cell. In the two dimensional CA this is also common such that it forms a torus structure.

4.1 Majority problem

The Majority Problem can be defined as follows: *Given a set* $A = \{a_1, ..., a_n\}$ *with* n *odd and* $a_m \in \{0, 1\}$ *for all* $1 \leq m \leq n$*, answer the question: 'Are there more ones than zeros in* A*?'.*

The Majority Problem first does not seem to be a very difficult problem to solve. It seems only a matter of counting the ones in the set and then comparing them to the number of zeros. Yet, when this problem is assigned to a CA it becomes a lot more difficult. This is because the rule in a CA does not let a position look past its neighborhood and that is why the cells all have to work together and use some form of communication.

Given that the relative number of ones in C^0 is written as λ, in a simple binary CA the Majority Problem can be defined as: *Find a rule that, given an initial state of a CA with* N *odd and a finite number* I *of iterations to run, will result in an 'all zero' state if* $\lambda < 0.5$ *and an 'all one' state otherwise.*

The fitness (f) of a rule is therefore defined as the relative number of correct answers to 100 randomly chosen initial states, where a 'correct answer' corresponds to an 'all zero' state if $\lambda < 0.5$ and an 'all one' state otherwise. In [10] the authors found that using a uniform distribution over λ for the initial states enhanced performance greatly; this is used here as well. The best runs will be tested using randomly chosen initial states with a normal distribution over the

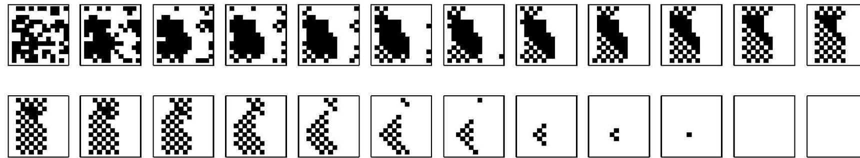

Fig. 2. This figure shows a correct classification of the Majority Problem by a two dimensional CA with both width and height equal to 13 and $\lambda = 84/169$. The transition rule was one of the best tested in the experiment and scored $F_{169,10^3} = 0.715$.

number of ones. The relative number of correct classifications on these states is written as $F_{n,m}$ where n is the width of the CA and m is the number of tests conducted.

Preliminary experiments showed that it took much more time to evolve rules for the Moore neighborhood than for the von Neumann neighborhood. The tests that were done with the Moore neighborhood also did not result in any encouraging results, this being in line with [7]. That is why the von Neumann neighborhood was chosen for this experiment. Because this neighborhood consists of five positions, the search space for CA rules is a lot smaller than in the one dimensional experiment. Instead of the $2^7 = 128$ bits in the rule, R now consists of $2^5 = 32$ bits, thus drastically decreasing the search space. This means that the search space decreased from 2^{128} to 2^{32} and is now $2^{(128-32)} = 2^{96}$ times smaller!

For this experiment we used a CA with width = 13 and height = 13. This means that these CA have $13 \times 13 = 169$ cells (N) and are $169 - 149 = 20$ cells larger than the one dimensional CA used in the original experiment.

This algorithm was run 300 times and each winning rule was tested by calculating $F_{N,M}$ using $F_{169,10^3}$. These results are plotted against results of our own one dimensional experiment (not reported here, analogue to [10, 11]) in Figure 1. The striking difference between this distribution of fitness and the distribution of fitness in the one dimensional experiment is the absence of the peak around $F_{N,M} \approx 0.5$ in the two dimensional results. In those results almost all the evolved rules have a fitness above 0.58. A fitness around 0.66 seems to be average and the best rules have a fitness above 0.7. That is all very surprising taking into account that the von Neumann neighborhood only consists of 5 cells.

The Majority Problem is a good example of a problem that forces cells in a CA to 'communicate' with each other. The communication 'particles' can be seen in the one dimensional experiment, but are not easily spotted in the two dimensional experiment. That does not mean there are no 'particles' traveling in the two dimensional CA, because it might be very hard to identify these particles. In a two dimensional CA 'particles' are no longer restricted to traveling in only one direction, but can travel to multiple directions at the same time. Traveling particles in two dimensional CA can therefore look like expanding areas with a distinct border. But there might be multiple particles traveling at

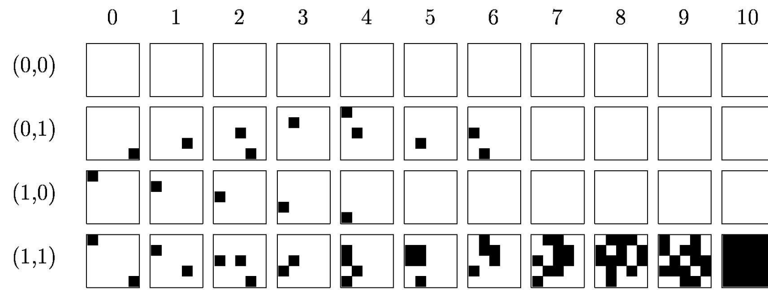

Fig. 3. This figure displays the iterations of a CA solving the AND problem. Every row shows the iteration of the rule using a different initial state. Note that in the first column ($t = 0$) the initial states are clearly visible and in the last column the coloring matches the output of an AND port.

the same time, meeting each other and thereby creating new particles. This is why communication between cells in a two dimensional CA is not very visible in the Majority Problem, although results show that this communication is present.

4.2 AND and XOR Problem

To show the communication between cells in a two dimensional CA a different experiment was conducted. A genetic algorithm was used to evolve rules for two dimensional CA that could solve the simple binary operators AND and XOR. These operators both have two input values and one output value which can only be determined if both input values are known. This is unlike the OR operator for example where the output value is always one if one or more of the input values is one, so if only one input value is known to be one then the value of the other input value is not needed. This may look very trivial, but it is very important in order to force the CA to combine the two values and thereby communicate.

The AND Problem To show the communications in a CA the information that needs to be combined must be initialized as far apart as possible. The following problem definition takes this into account: *Given a square CA with two 'input cells', one top left and one bottom right: find a rule that iterates the CA so that after I iterations the CA is in an 'all one' state if both the 'input cells' were one in the initial state and in an 'all zero' state otherwise.*

Small two dimensional CA were used with a width and a height of 5 cells and I was set to 10. The borders of the CA were unconnected to allow a larger virtual distance between the two corner cells. This means that the leftmost cell in a row was not connected to the rightmost cell in the same row and the topmost cell was not connected to the bottommost cell as was done with the Majority Problem experiment. Instead every cell on the border of the CA was connected to so called 'zero-cells'. These 'zero-cells' stay zero whatever happens.

Table 1. Fitness values found in the AND problem.

	Number of runs			
	Neumann		*Moore*	
Fitness	*with crossover*	*without crossover*	*with crossover*	*without crossover*
100	0	0	31	21
98-99	0	0	41	54
95-97	0	0	14	25
90-94	77	93	14	0
80-89	23	7	0	0
70-79	0	0	0	0
< 70	0	0	0	0

When using two input cells, there are four different initial states. These states are written as $S_{(v_1,v_2)}$ where v_1 and v_2 are the two input values. All cells other than the two input cells are initialized with zero.

The fitness of a rule is defined as the total number of cells that have the correct values after I iterations. The number of ones in iteration t is written as $O^t_{(v_1,v_2)}$. The total fitness of the AND problem is defined as $f = (N - O^I_{(0,0)}) + (N - O^I_{(0,1)}) + (N - O^I_{(1,0)}) + O^I_{(1,1)}$. This makes the maximum fitness equal to $4 \times 5 \times 5 = 100$.

In this experiment another variation of the simple genetic algorithm was used. A generation step starts by sorting the rules according to their fitness. Then it selects the top ten% of the rules as 'elite' rules and copies them without changes to the next generation. Every 'elite' rule is then copied nine times or is used in single-point crossover to make the other 90% of the population. Both methods were tested and compared. The generated rules are mutated and also moved to the next generation. Mutation is done by flipping every bit in the rule with a probability p_m. The algorithm stops if it finds a rule with $f = 100$ or it reaches 1000 generations. In preliminary experiments a number of different values of p_m were tested. Setting p_m to a rather high value of 0.05 turned out to be the most effective choice, confirming our insight that with increasing selection strength higher mutation rates than the usual $\frac{1}{l}$ (l being the the length of the binary string) are performing better [1].

The algorithm was run 100 runs with and without single-point crossover and using both the von Neumann and the Moore neighborhoods. The results are shown in Table 1.

Although rules evolved with the von Neumann neighborhood are not able to solve the problem perfectly, it is already surprising that it finds rules which work for 93%, for such a rule only misplaces 7 cells in the final state. All the other 93 cells have the right value. This suggests that the information was combined, but the rule could not fill or empty the whole square using the same logic.

The Moore neighborhood is clearly more powerful and was able to solve the problem perfectly. The rules that are able to do this clearly show commu-

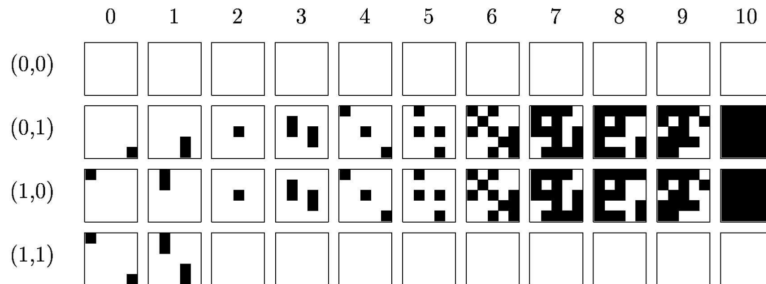

Fig. 4. This figure displays the iterations of a CA solving the XOR problem. Every row shows the iteration of the rule using a different initial state. Note that in the first column ($t = 0$) the initial states are clearly visible and in the last column the coloring matches the output of an XOR port.

nicational behavior in the form of "traveling" information and processing this information at points where information "particles" meet.

It is also surprising that using crossover in combination with a Neumann neighborhood does not outperform the same algorithm without the crossover. This may be due to the order of the bits in the transition rule and their meaning. This is worth exploring in future work. Maybe using other forms of crossover might give better results in combination with multi dimensional CA.

The XOR Problem The XOR Problem is not much different from the AND problem. We used the same genetic algorithm and the same CA setup. The only difference is the fitness function. We defined the XOR problem as follows: *Given a square CA with two 'input cells', one top left and one bottom right: find a rule that iterates the CA so that after I iterations the CA is in an 'all one' state if only one of the 'input cells' was one in the initial state and in an 'all zero' state otherwise.* This means that the total fitness of the XOR problem is defined as $f = (N - O^I_{(0,0)}) + O^I_{(0,1)} + O^I_{(1,0)} + (N - O^I_{(1,1)})$.

The algorithm was run with $p_m = 0.05$ for a maximum of 1000 generations for 100 runs with both Neumann and Moore neighborhoods with and without single point crossover. The results are shown in Table 2.

These results support earlier finding in suggesting that single-point crossover doesn't really improve the performance when used in a two dimensional CA. The results show that the algorithm using only mutation has found ways to solve this rather difficult communicational problem. The Neumann neighborhood seemed unable to perform for 100%, yet it came rather close with one rule classifying the problem for 92%. The algorithm found one transition rule using the Moore neighborhood that is able to solve the problem for the full 100%. This rule depicted in Figure 4 shows clear signs of "traveling particles" and is another example of how a local rule can trigger global behavior.

Table 2. Fitness values found in the XOR problem.

	Number of runs			
	Neumann		*Moore*	
Fitness	*with crossover*	*without crossover*	*with crossover*	*without crossover*
100	0	0	0	1
98-99	0	0	4	4
95-97	0	0	7	6
90-94	2	1	19	21
80-89	76	96	69	66
70-79	18	3	1	2
< 70	4	0	0	0

5 Evolving bitmaps

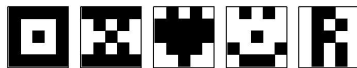

Fig. 5. The bitmaps used in the pattern generation experiment.

Now that it is shown that two dimensional CA can communicate, it is time to increase the challenge for the CA a bit. The aim of this experiment is to evolve rules for two dimensional CA that generate patterns (or bitmaps).

The Bitmap Problem is defined as follows: *Given an initial state and a specific desired end state: find a rule that iterates from the initial state to the desired state in less than I iterations.* Note that this does not require the number of iterations between the initial and the desired state to be fixed.

The CA used in this experiment is not very different from the one used in the AND/XOR experiment (section 4.2). In preliminary experiments we tried different sizes of CA, but decided to concentrate on small square bitmaps with a width and a height of 5 cells (as done in section 4.2). To make the problem harder and to stay in line with earlier experiments the CA have unconnected borders like in section 4.2. The von Neumann neighborhood was chosen instead of the Moore neighborhood and therefore s_n consist of 5 cells ($r = 1$) and a rule can be described with $2^5 = 32$ bits. The search space therefore is $2^{32} = 4294967296$.

After testing different initial states, the 'single seed' state was chosen and defined as the state in which all the positions in the CA are zero except the position ($\lfloor \text{width}/2 \rfloor, \lfloor \text{height}/2 \rfloor$) which is one. For the GA we used the same algorithm as we used in the AND and XOR experiments. Because this experiment uses a Neumann neighborhood and the AND and XOR experiments suggested that the combination between the von Neumann neighborhood and single point crossover was not a good idea, this experiment used only mutation. Like in sec-

tion 4.2 mutation is performed by flipping every bit in the rule with a probability p_m. In this experiment $p_m = 1/32 = 0.03125$.

In trying to be as diverse as possible five totally different bitmaps were chosen, they are shown in Figure 5. The algorithm was run 100 times for every bitmap for a maximum of 5000 generations. The algorithm was able to find a rule for all the bitmaps, but some bitmaps seemed a bit more difficult than others. Table 3 shows the number of successful rules for every bitmap. Note that symmetrical bitmaps seem to be easier to generate then asymmetric ones.

Table 3. Number of successful rules found per bitmap.

Bitmap	*Successful rules (out of a 100)*
"square"	80
"hourglass"	77
"heart"	35
"smiley"	7
"letter"	9

Although this experiment is fairly simple, it does show that a GA can be used to evolve transition rules in two dimensional CA that are able to generate patterns even with a simple von Neumann neighborhood. Ongoing experiments with bigger CA suggest that they don't differ much from these small ones, although the restrictions on what can be generated from a single-seed state using only a von Neumann neighborhood seem to be bigger when size of the CA increases.

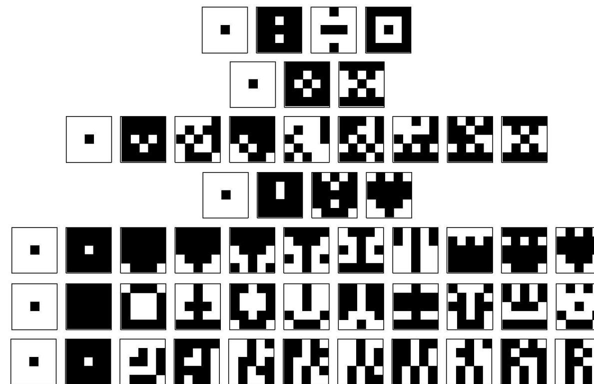

Fig. 6. This figure shows some iteration paths of successful transition rules.

6 Conclusions

The aim of the experiment reported in this paper was to demonstrate the capability of evolutionary algorithms, here a fairly standard genetic algorithm, to parameterize other methods such as, specifically, cellular automata. From the experimental results reported, one can conclude that this kind of inverse design of CA is possible by means of evolutionary computation in a clear, straightforward, and very powerful way. The results clearly indicate that real world applications of CA could also be tackled by this approach, and the unconventional programming of CA by means of EA's is not only a possibility, but a useful and efficient method to parameterize this kind of algorithm.

References

1. Th. Bäck, D. B. Fogel, and editors Michalewicz, Z., editors. *Handbook of Evolutionary Computation*. Oxford University Press and Institute of Physics Publishing, Bristol/New York, 1997.
2. D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, New York, 1995.
3. L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons, 1966.
4. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
5. D. E. Goldberg. *The Design of Invocation:Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.
6. J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
7. S. Inverso, D. Kunkle, and C. Merrigan. Evolutionary methods for 2-d cellular automata computation. www.cs.rit.edu/˜drk4633/mypapers/gacaProj.pdf, 2002.
8. J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, 1992.
9. J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.
10. M. Mitchell and J.P. Crutchfield. The evolution of emergent computation. Technical report, Proceedings of the National Academy of Sciences, SFI Technical Report 94-03-012, 1994.
11. M. Mitchell, J.P. Crutchfield, and P.T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.
12. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
13. I. Rechenberg. *Evolutionsstrategie '94*. Fromman-Holzboog Verlag, Stuttgart, 1994.
14. H. P. Schwefel. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. *Interdisciplinary Systems Research*, 26, 1977.
15. H. P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.
16. S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983.