Using a Genetic Algorithm to Evolve Behavior in Multi Dimensional Cellular Automata

Emergence of Behavior

R. Breukelaar
Leiden Institute of Advanced Computer Science,
Universiteit Leiden
P.O. Box 9512, 2300 RA Leiden, The Netherlands
rbreukel@liacs.nl

Th. Bäck
Leiden Institute of Advanced Computer Science,
Universiteit Leiden
P.O. Box 9512, 2300 RA Leiden, The Netherlands
and
Nutech Solutions GmbH,
Martin-Schmeisser-Weg, 44227 Dortmund, Germany
baeck@liacs.nl

ABSTRACT

Cellular automata are used in many fields to generate a global behavior with local rules. Finding the rules that display a desired behavior can be a hard task especially in real world problems. This paper proposes an improved approach to generate these transition rules for multi dimensional cellular automata using a genetic algorithm, thus giving a generic way to evolve global behavior with local rules, thereby mimicking nature. Three different problems are solved using multi dimensional topologies of cellular automata to show robustness, flexibility and potential. The results suggest that using multiple dimensions makes it easier to evolve desired behavior and that combining genetic algorithms with multi dimensional cellular automata is a very powerful way to evolve very diverse behavior and has great potential for real world problems.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE—*Problem Solving, Control Methods, and Search*

General Terms

Algorithms, Experimentation, Performance, Theory

1. INTRODUCTION

Science has long since been fascinated by how simple localized behavior can produce very complex global behavior. Well known examples can for instance be found in nature in the form of ant colonies. Ants follow a few very simple rules that don't seem to be very intelligent or complex at the level of a single ant. Yet the behavior and structure of an entire ant colony seems very complex and organized. Ants seem to work together to collect food, defend the colony and attack the enemy. This paper will not go into the behavior of ants or any other specific species, but will rather use a far more abstract representation of individuals and their behavior called Cellular Automata (CA).

In nature behavior can often be contributed to evolution. Especially in small insects like ants the behavior seems only to have a genetic origin. This implies that the organization and communication in an ant colony could also be contributed to evolution. This paper reports how the same genetic principles of evolution can be applied to CA to evolve a desired global behavior with only local rules. Earlier work [9, 10, 8] has shown that using evolution on one dimensional CA can be a very powerful combination. This paper suggests an improved Genetic Algorithm (GA) and more importantly shows how this algorithm can be applied to multi dimensional CA. By evolving behavior in a multi dimensional space the hope is that the approach could be applied to real world problems in areas like robotics, parallel computing, biological and social modeling, artificial intelligence and image analysis. CA are already used in these areas and designing these models is often very time consuming, therefore automating this process will be a welcome application.

The paper starts by giving a brief introduction to CA, first the one dimensional case, followed by a generic way to extend this to multiple dimensions. Next it describes an approach using a generic GA to evolve the behavior in the CA. This approach is then used in three different experiments that show its potential and flexibility.

2. CELLULAR AUTOMATA

According to [11] 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

GECCO'05, June 25–29, 2005, Washington, DC, USA.

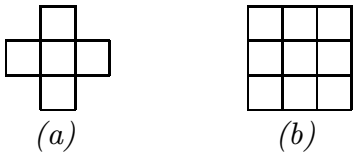

Figure 1: Two often used and well known two dimensional neighborhoods. *(a)* the von Neumann neighborhood and *(b)* the Moore neighborhood.

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 time step (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.

2.1 Multi Dimensional Cellular Automata

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.

The most simple two dimensional CA can be viewed as a grid of positions $a(i,j)$ instead of a row in the one dimensional case. The borders of this CA are connected in such a way that every first cell in a row $a(1,j)$ is connected to the last cell $a(w,j)$ and every first cell in a column $a(i,1)$ is connected the the last cell in that column $a(i,h)$. This topology is also known as a 'torus' or 'donut' shape.

There are two neighborhoods that are often used in this two dimensional space being the von Neumann neighborhood and the Moore neighborhood (Figure 1). These neighborhoods can be extended to have a larger radius and more dimensions if defined in terms of distance: Every cell in a neighborhood has a path to the center cell that is equal or less than r steps to 'adjacent' cells. In a CA with d dimensions $e_1, e_2, ..., e_d$ cells $a(i_1, i_2, ..., i_d)$ and $b(j_1, j_2, ..., j_d)$ are 'adjacent' in a von Neumann neighborhood if

$$\sum_{k=1}^{d} \min(|i_k - j_k|, e_k - |i_k - j_k|) = 1$$

In a Moore neighborhood cells are 'adjacent' if

$$\min(|i_k - j_k|, e_k - |i_k - j_k|) \leq 1 \text{ for } 1 \leq k \leq d$$

Note that a one dimensional von Neumann neighborhood is equal to a one dimensional Moore neighborhood.

Transition rules are defined in the same way as in the one dimensional CA where every bit in the index of the bitstring represents one input cell in the neighborhood. The cells in the neighborhood are numbered from 1 to n in a recursive way over the d dimensions. Note that this means that the center cell is always numbered $\frac{n+1}{2}$.

The number of cells in these neighborhoods grows very fast if r or d is increased. Table 1 clearly shows that the Moore neighborhood grows a lot faster than the von Neumann neighborhood. In this paper we will use a few different combinations and explore their differences.

	r						
	0	1	2	3	4	5	6
$S^N(1,r)$	1	3	5	7	9	11	13
$S^N(2,r)$	1	5	13	25	41	61	85
$S^N(3,r)$	1	7	25	63	129	231	377
$S^N(4,r)$	1	9	41	129	321	681	1289
$S^N(5,r)$	1	11	61	231	681	1683	3653
$S^N(6,r)$	1	13	85	377	1289	3653	8989
$S^M(1,r)$	1	3	5	7	9	11	13
$S^M(2,r)$	1	9	25	49	81	121	169
$S^M(3,r)$	1	27	125	343	729	1331	2197
$S^M(4,r)$	1	81	625	2401	6561	14641	28561
$S^M(5,r)$	1	243	3125	16807	59049	161051	371293
$S^M(6,r)$	1	729	15625	117649	531441	1771561	4826809

Table 1: The number of cells in neighborhoods in multi dimensional CA. $S^N(d,r)$ stands for a d dimensional von Neumann neighborhood with a radius r and $S^M(d,r)$ represents a d dimensional Moore neighborhood with radius r. Note that $S^N(d,r)$ is a lot smaller and symmetric.

3. GENETIC ALGORITHM

Genetic algorithms (GAs) are often used to find solutions in large discrete search spaces that are to big to iterate completely [4, 5, 6]. From an evolutionary point of view, the fact that CA define behavior in the form of a binary transition rule which can easily be evolved with a GA is a bonus. M. Mitchell, J. P. Crutchfield and P. T. Hraber have shown [9, 10] that using a simple GA to evolve transition rules for the majority problem (explained in section 4) can already give surprisingly good results. Other approaches have been proposed by for instance S. Inverso, D. Kunkle and C. Merrigan [7]. In this paper we will suggest a different GA that outperforms used approaches and test its robustness by running different experiments with different radius r and dimension d settings.

The GA in this paper uses tournament selection as defined in [1]. This selection involves running 'tournaments' on the

population in order to determine the next generation. Every tournament q individuals are selected at random from generation t and the one with the highest fitness is then copied to generation $t+1$. For a population of λ individuals this process is repeated λ times and the result is generation $t+1$ with λ individuals.

After selection generation $t+1$ is mutated. Mutation is done by using single-point crossover on a subset of the population and mutating the resulting individuals using probabilistic bit flip. The relative number of individuals that are mutated in this way is denoted by the crossover-rate c. Mutation is done by flipping every bit in the individual with a probability m.

All the individuals in the pool are initialized at random with a normal distribution over the number of ones in an individual. This means that the number of individuals with a certain number of ones will be roughly equal to the number of individuals with another. This prevents the algorithm from specializing in a particular area of the search space at the beginning of the algorithm. The evolution ends after D generations and the best individual of the last generation is considered to be the answer.

The GA is expected to behave differently with different settings of q, c, m and D, but it should ideally be usable in a large range of different experiments without much changes. The majority problem was used to determine some good settings and then these settings were used in the other experiments without any changes. It is expected that the algorithm is capable of better results if the setting-space is explored more intensely. This will take a long time though and is fuel for further work.

4. MAJORITY PROBLEM

The majority problem is often used to show the power of cellular automata. Its a simple example of how local rules can perform a global task and is often studied in the one dimensional form. 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 at first glance 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 converted to the dimensions of a CA and the restriction of local rules 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 transition rule that, given an initial state of a CA with N odd and a finite number of iterations to run (I), will result in an 'all zero' state if $\lambda < 0.5$ and an 'all one' state otherwise. The 'all zero' state being the state in which every cell in the CA is zero and the 'all one' state being a the state in which every cell is one.

Evaluating a transition rule for this problem is done by iterating M randomly generated initial states and calculating the relative number of correct classification. The fitness of a transition rule is denoted with $F_{N,M}$ where N is the width

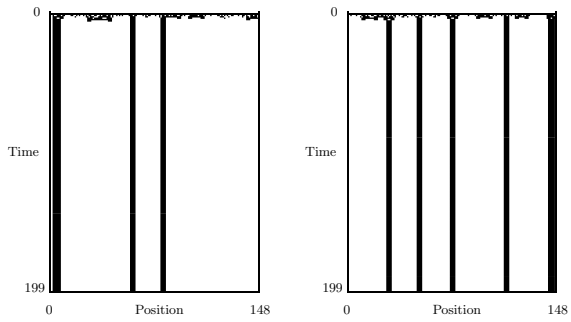

Figure 2: These are examples of majority problem classification by the "majority rule". The pictures show how the rule gets stuck on "thick lines" in the time plot. Time t proceeds from top to bottom and every row corresponds to C^t.

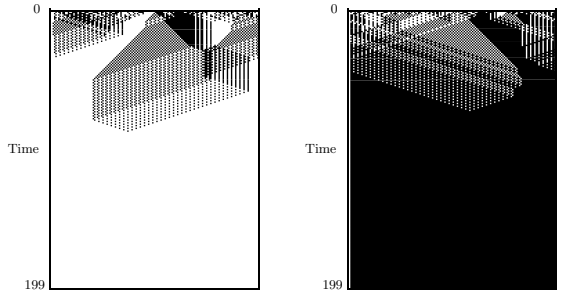

Figure 3: These are examples of majority problem classification by the rule found by David, Forrest and Koza. Both are correct classifications (a) with 74 ones in the initial state, (b) with 75.

of the CA. The fitness can be calculated with different distributions over the number of ones in the initial state, but the default is a binomial distribution (denoted with $F^B_{N,M}$) where every cell in the CA has a 50% chance of being initiated with a one or a zero for every initial state.

The first intuitive rule to come up with is often called the 'majority rule'. This being the rule where the output value is 1 if the number of ones in the neighborhood is more than the number of zeros, and a zero otherwise. Surprising as it may seem this does not at all solve the problem (as is shown in figure 2). The majority rule fails to work on boundaries of thick lines in the time plot. There the cells can't "agree" on the global answer. The cell just left of such a thick line is zero and because all other cells left of it in the neighborhood are also zero, it "decides" to stay that way. Yet its neighbor to the right is one and sees only ones on its right and therefore decides to stay one. This way the information fails to propagate through the CA and classification is incorrect.

A lot of people have come up with different rules to solve this problem, one such rule is the GKL rule after Gacs, Kurdyumov and Levin [3]. This rule is good at classifying the majority problem and does it for 81.6% of the test cases with a width of 149 cells. For 17 years this was the best rule and then Lawrence Davis found a better one in 1995 which

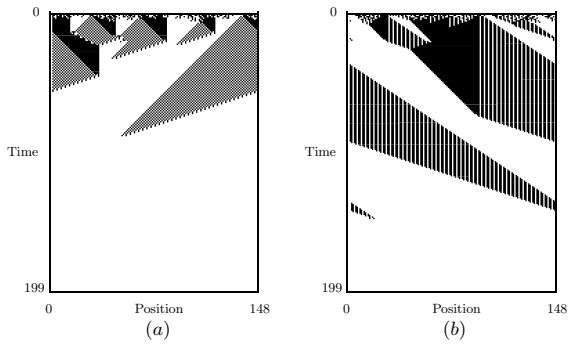

Figure 4: This figure displays two correct classification of the majority problem by two different particle based rules generated with the GA. (a) **has** $F^B_{149,10^4} \approx 0.76$ **and** (b) **has** $F^B_{149,10^4} \approx 0.75$**. with** $N = 149$**.**

did 81.8%. In the same year Rajarshi Das found a rule that did 82.178%. Then in 1996 David, Forrest and Koza found a rule by cleverly using genetic programming that was able to classify 82.326% correctly [2].

Although these rules are very impressive it is believed that there is no definite solution for the problem as long as the neighborhood is smaller than the size of the CA. It is already a big accomplishment for a CA to get 70% of all random initial states correct, for this shows there is some kind of communication going on; some kind of emerging behavior.

4.1 Genetic Algorithm

The algorithm as proposed in section 3 was used to evolve transition rules for this problem. Different parameter settings were tried to find optimal setting and to examine the robustness of the algorithm. The experiment started out with one dimensional CA with $N = 149$ and $I = 320$ for historical reasons [9, 10, 8, 12]. Initial parameters for the GA were $\lambda = 100$, $q = 10$, $c = 0.10$, $m = \frac{2}{2^S} = \frac{2}{128} = 0.015625$ and $D = 100$.

Preliminary experiments as well as experiments by Packard et al. [8] and Mitchell et al. [9, 10] suggest that it is very difficult to evolve good transition rules with a GA while using a binomial distribution over the number of ones in the initial states. Note that with this distribution the number initial states with $\lambda \approx 0.5$ which are the hardest to classify is very high. The solution for this is using a uniform distribution while evolving the rules. This distribution generates more 'easy' initial states with a large difference between the number of ones and the number of zeros, thus making it easier to train the desired behavior. The fitness using initial states with this uniform distribution over the number of ones is denoted with $F^U_{N,M}$.

This distribution has a drawback though. Because rules are selected using a different fitness function than the one used to test them in the end, it seems possible that the rules will specialize in a behavior that would seem good for the uniform distribution, but very bad for the binomial distribution. To counter this effect a "gliding distribution" is used. This distribution is different for every generation of the genetic algorithm. It "glides" gently from an uniform distribution in generation 0 to a binomial distribution in generation D. This is achieved by generating $\lfloor M \cdot \frac{g}{D} \rfloor$ initial states with a binomial distribution and $\lfloor M \cdot (1 - \frac{g}{D}) \rfloor$ initial states with a uniform distribution, where g is the current generation. This distribution has the benefits of the uniform distribution in the beginning of the algorithm without the drawbacks at the end. The fitness using this distribution is denoted with $F^G_{N,M,g}$. Note that $F^G_{N,M,0} = F^U_{N,M}$ and $F^G_{N,M,D} = F^B_{N,M}$.

$F^B_{149,10^4}$	q					
	2	3	5	10	20	50
0.0 - 0.5	23	0	1	0	2	3
0.5 - 0.55	37	0	0	0	2	2
0.55 - 0.6	14	6	4	1	2	4
0.6 - 0.65	25	79	69	48	52	47
0.65 - 0.7	1	16	42	50	39	40
0.7 - 0.75	0	0	0	1	3	4
0.75 - 0.8	0	0	0	0	0	0
0.8 - 1.0	0	0	0	0	0	0

Table 2: This table shows the fitness distribution using different values for the tournament size q. Other settings are the same as the initial values proposed in section 4.1.

4.2 Results

Different parameter settings were tried on the GA. Experiments with different values for the mutation rate m did not show any real improvement and it was concluded that $m = \frac{2}{128} = 0.015625$ was a good setting. Also changing the number of generations D did not seem to yield improvements immediately although in theory a larger D should increase the chance of good results. Because of the time restrictions and historical compatibility with [9, 10] $D = 100$.

Exploring different tournament sizes q values however seemed to give very different results. Six different experiments were conducted using $q = \{2, 3, 5, 10, 20, 50\}$. Each setting was run a 100 times. Results are shown in Table 2. Note that these results imply that a high selection pressure is needed to gain good results. Settings $q = \{2, 3, 5\}$ do not seem to be very good in generating rules that exceed the 0.7 barrier, $q = 10$ is better, but $q = \{20, 50\}$ generate both very good results. Because $q = 50$ seems to produce more 'bad' rules with $F < 0.6$, therefore it was decided to use $q = 20$ in the future.

Different crossover rates also seemed to change the results. Using the new tournament size $q = 20$ four different values for c were tried: 0.6, 0.8, 0.9 and 0.95. Table 3 shows the results. Note that the best results are achieved using $c = 0.6$, but the difference is minimal. This together with the findings for the different mutation rates m implies that the algorithm is robust under different mutation settings and might be usable for different problems without changing these settings.

Next this same algorithm was used to evolve transition rules for CA with multiple dimensions. Instead of using a one dimensional neighborhood with $r = 3$ the von Neumann neighborhood with $r = 1$ was used with $d = \{2, 3\}$. Note that $S^N_{2,2} = 5$, that is two cells less than the one dimensional experiment and that means a bit string of $2^5 = 32$ bits instead of 128 bits. In theory this would imply less complex rules and worse performance. The algorithm was run a 100 times with $d = 2$ and $d = 3$. Because a two dimensional

$F^B_{149,10^4}$	c			
	0.6	0.8	0.9	0.95
0.0 - 0.5	0	0	2	2
0.5 - 0.55	2	1	2	0
0.55 - 0.6	1	1	2	2
0.6 - 0.65	54	52	52	56
0.65 - 0.7	37	42	39	36
0.7 - 0.75	3	4	3	3
0.75 - 0.8	3	0	0	1
0.8 - 1.0	0	0	0	0

Table 3: This table shows the fitness distribution using different values for the crossover rates c, $q = 20$ and other settings are the same as the initial values proposed in section 4.1.

$F^B_{N,10^4}$	d		
	1	2	3
0.0 - 0.5	0	0	1
0.5 - 0.55	2	0	0
0.55 - 0.6	1	4	0
0.6 - 0.65	54	17	14
0.65 - 0.7	37	72	55
0.7 - 0.75	3	1	30
0.75 - 0.8	3	0	0
0.8 - 1.0	0	0	0

Table 4: This table shows the fitness distribution using different number of dimensions d. Note that $N = 149$ for $d = 1$, $N = 169$ for $d = 2$ and $N = 343$ for $d = 3$. $q = 20$, $c = 0.6$ and other settings are the same as the initial values proposed in section 4.1.

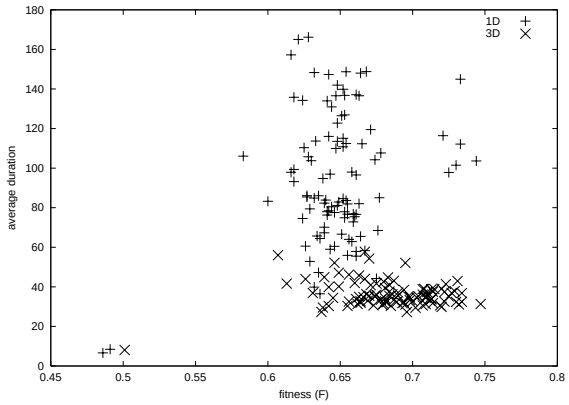

Figure 5: This figure plots the average duration of the evolved rules against the fitness of the rules. It compares the one dimensional with the three dimensional topology and shows that a three dimensional topology has a much smaller average duration, but similar fitness. Note that the two dimensional rules are omitted to make the plot readable.

CA is square and must have an odd number of cells the dimensions were set to 13×13, this means the CA consist of 169 cells. For the three dimensional CA it was even harder, because the nearest, bigger dimension with an odd number of cells is $7 \times 7 \times 7$, resulting in 343 cells. That is why the results in Table 4 are so surprising. They show how the GA did not have any problem finding rules for the two dimensional CA that performed with similar fitness on a larger CA. Note that the number of rules with $F > 0.65$ is even higher in the two dimensional case. With $d = 3$ the results are even more compelling. The number of rules found with $F > 0.7$ is a lot higher than in the one dimensional experiment. This suggest that using a topology with multiple dimensions makes it easier to solve the majority problem.

The average duration of a transition rule is defined as the average number of iterations a rule needs to result in a fixed state. Note that this does not have to be the desired state. The time it takes for a CA to perform its task is of great importance in real world problems. The quicker the rule, the better the solution. Figure 5 shows the average durations for every evolved rule in the one dimensional and three dimensional experiment plotted against their fitness. Note that the average duration of the three dimensional rules is a lot lower than the duration of the one dimensional rules, especially for the best rules around $F \approx 0.73$. This suggests that using multiple dimensions is a lot more efficient than using only one dimension. The two dimensional rules are omitted from the plot to make the plot readable, but with most of the average durations of the two dimensional experiment between 20 and 60, and the best rules with a duration of around 50, the theory is supported.

5. CHECKERBOARD PROBLEM

Another often presented CA problem is the "checkerboard problem". This problem demonstrates how a CA can generate a simple global pattern using only local rules. The checkerboard problem can be defined as follows:

Find a transition rule that, given an initial state of a CA, iterates this CA to a "checkerboard pattern" within I iterations.

A checkerboard pattern can be defined as a state in which directly adjacent cells in a CA have different values. Such a pattern could look like: $\{0, 1, 0, 1, ..., 1, 0, 1\}$ in a one dimensional CA. Note that the first cell and the last cell are linked and should therefore also have different values. The problem is more intuitive in a two dimensional CA where cells are not only connected horizontally, but also vertically and the desired state therefore resembles a checkerboard. The problem can even be imagined in three dimensional CA where the end result should resemble a stack of checkerboards where every odd board in the stack is turned 90 degrees. In theory this problem is extendable to more dimensional spaces, but only CA with $d = \{1, 2, 3\}$ will be considered here.

Just like in the majority problem the checkerboard problem used multiple initial states to determine the fitness of a transition rule. These initial states however don't need to use the gliding distribution used in the majority problem, because preliminary experiments showed that there is already evolution with a binomial distribution. The fitness of a transition rule is measured in the relative number of directly adjacent cells in the end state that have an inverted value.

The same GA was used as in the majority problem, even

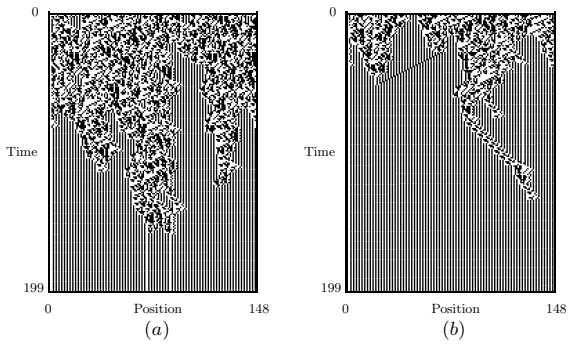

Figure 6: This figure shows two one dimensional CA iteration for the checkerboard problem. Note that (a) does not result in a perfect pattern, whereas (b) does.

the parameters have the same values (that is optimal values as used in the last experiments). That means: $q = 20$, $c = 0.6$, $m = \frac{2}{2^5}$ and $D = 100$. The number of cells in a dimension of the CA needs to be even, else a perfect checkerboard pattern will be impossible. CA with 150, 12^2 and 6^3 cells were used for $d = 1, 2$ and 3 respectively.

5.1 Results

The algorithm was run 100 times for all three topologies also used in the majority problem: the one dimensional CA with $r = 3$, the two dimensional CA with a von Neumann neighborhood with $r = 1$ and the three dimensional CA also with a von Neumann neighborhood with $r = 1$. For all three runs the same parameters were used as in the last experiments on the majority problem. Note that m has different values for different topologies, because this variable is dependent on the number of cells in a neighborhood S and was always set to $\frac{2}{2^S}$. This means that for the one and three dimensional CA $m = \frac{2}{2^7} = \frac{2}{128} = 0.015625$ and for the two dimensional CA $m = \frac{2}{2^5} = \frac{2}{32} = 0.625$.

In the one dimensional experiment all the runs resulted in transition rules with $F^B_{150,10^3} > 0.95$ and the best rule had a fitness of 0.998. The two dimensional experiment had similar results with about 80% of the runs with $F^B_{12^2,10^3} > 0.95$ and the best rule with a fitness $F^B_{12^2,10^3} = 0.994$. Note that these results are achieved with a lot smaller neighborhood. In the three dimensional experiment however all the runs (except two) evolved rules with fitness values $F^B_{6^3,10^3} > 0.996$. Some of the rules even registered a perfect fitness for all the 10^3 random initial states. After some tests it seemed that the best rules in the three dimensional experiment had a fitness of $F^B_{6^3,10^5} \approx 0.99997$ were the best one dimensional rule scored $F^B_{150,10^5} \approx 0.99834$.

The average duration was calculated for every run like it was done in the majority problem. Figure 8 shows the results. Note that the two and three dimensional rules have a lot smaller duration and a far better fitness distribution than the one dimensional rules. These results support earlier findings in the majority problem that suggest that multi dimensional CA can perform a task much faster and are easier to evolve with a GA.

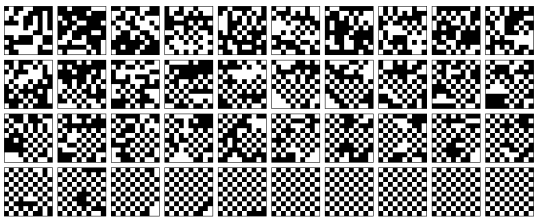

Figure 7: This figure shows a correct two dimensional CA iteration for the checkerboard problem. It start top-left with a random initialization of a 10×10 CA, iterates from left to right, top to bottom and ends up with a perfect checkerboard pattern in the end state.

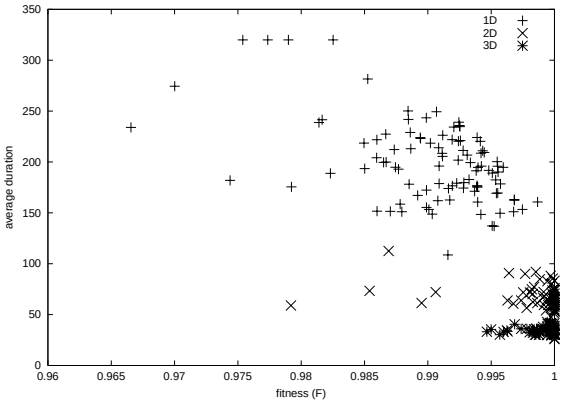

Figure 8: This figure plots the average duration against the fitness for every run and compares the one, two and three dimensional experiments. Notice how not only the fitness distribution improves, but also the average duration decreases if the d is increased.

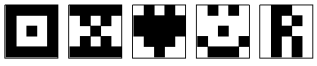

Figure 9: The bitmaps used in the pattern generation experiment.

Table 5: Number of successful rules found per bitmap.

Bitmap	*Successful rules (out of a 100)*
"square"	100
"hourglass"	96
"heart"	55
"smiley"	29
"letter"	18

6. EVOLVING BITMAPS

To test our approach on a more generic and complex problem than the previous to problems, the bitmap problem was used. 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 iteration between the initial and the desired state to be fixed and can be any number between 1 and I.

The CA used in this experiment is the same as the one used in the other two experiments. 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. The von Neumann neighborhood was chosen with $r = 1$ and therefore s_n consist of 5 cells and a rule can be described with $2^5 = 32$ bits. The search space therefore is 2^{32}. A bitmap of 32 b/w pixels would have the same number of possibilities, therefore this experiment is very challenging to say the least.

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. The CA was not expected to stay at the desired state as was expected in section 4. The neighborhood of a rule in the CA is very small and therefore it is already very difficult to make a rule stop altering the CA ones it has reached the desired state, but for a rule to go from the initial state to a desired state in the CA and then stay there is a bit to challenging.

For the GA the same parameters as for the other two experiments were used, that means $q = 20$, $c = 0.6$, $m = \frac{2}{2^S} = 0.625$. Only the number of generations for one run D was increased from 100 to 5000. Preliminary experiments had shown that convergence in this challenging experiment was a lot slower, but because there is only one initial state to check, computation time is lot faster and longer runs are possible.

In trying to be as diverse as possible five totally different bitmaps were chosen, they are shown in figure 9. The algorithm was run 100 times for every bitmap.

6.1 Results

The algorithm was able to find a rule for all the bitmaps, but some bitmaps seemed a bit more difficult than others. Table 5 shows the number of successful rules for every bitmap. Note that symmetrical bitmaps seem to be easier to generate than asymmetric ones. Figure 10 shows a few successful transition rules generated by the GA. Note that different transition rules can end up in the same desired state and have totally different iteration paths.

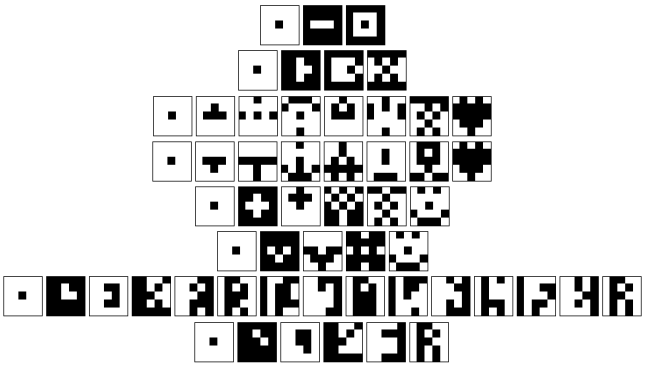

Figure 10: This figure shows some iteration paths of successful transition rules.

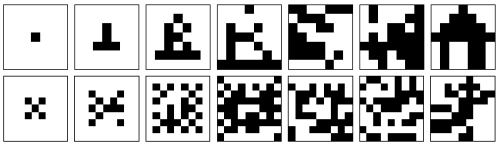

Figure 11: This figure shows two more iteration paths of transition rules evolved with the GA that use a Moore neighborhood with $r = 1$. The runs show that the GA is able to work with larger CA and bigger neighborhoods. The first image represents a house and the second a gecco (inspired by the logo of the GECCO conference).

To increase the challenge and test the scalability of this approach the size of the CA was increased. CA of 7×7 and 9×9 were tested using some bitmaps. After some preliminary experiments it seemed that the von Neumann neighborhood with $r = 1$ was having trouble generating the larger patterns. Note that a bitmap of 7×7 is almost twice big as a bitmap of 5×5 and has more cells than there are bits in a von Neumann rule. The Moore neighborhood with $r = 1$ however found rules that generated the bitmaps using the same algorithm as was done for all other experiments. Figure 11 shows some of these successful rules. Note that the last bitmap was inspired by the GECCO logo and contains no symmetry whatsoever. Although there is probably a limit to what bitmaps the Moore neighborhood can produce, these results suggests that this approach is not limited to CA size or neighborhood size.

Only one specific initial state was used in all the bitmap experiments. It is likely that this initial state was not the easiest initial state for iterating these different bitmaps. Other initial states might yield even better results. The experiments show the potential power of multi dimensional CA, the many different global behaviors that can result from one very simple CA and how an evolutionary approach can find local rules for a wide range of desired global behavior. Although in the bitmap problem only two dimensional CA were used, the results from the majority problem and the checkerboard problem suggest that this approach can easily be extended to CA with three or more dimensions and has the potential to generate transition rules for CA in real world problems.

7. CONCLUSIONS

The result show that the GA gives good results for three different experiments without changing the evolution parameters. The GA exceeds the performance found in [9, 10] and does this for different topologies of CA. It therefore shows to be a robust and flexible approach to evolve global behavior with local rules.

Furthermore the results suggest that multi dimensional CA can solve problems faster than one dimensional CA. Rules for these topologies can be evolved using exactly the same algorithm and have similar top fitness and better distribution of this fitness. Results therefore imply that it is easier to evolve rules for multi dimensional CA.

The results for the bitmap problem also imply that the global behavior of a CA is not limited to checkerboards and majority problems. This approach might be robust and flexible enough to be used in real world problems that are not easily described in terms of neighborhoods, like the bitmap problem for instance, and its simplicity might give an insight into the evolution of behavior.

8. ACKNOWLEDGMENTS

This work is part of the research programme of the 'Stichting voor Fundamenteel Onderzoek der Materie (FOM)', which is financially supported by the 'Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO)'.

9. 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] A. David, B. Forest, and H. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem, 1996.

[3] P. Gacs, G. L. Kurdyumov, and L. A. Levin. One dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsi*, 12:92–98, 1978.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[5] David 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] W. Li, N. H. Packard, and C. G. Langton. Transition phenomena in cellular automata rule space. *Physica D*, 45:77–94, 1990.

[9] 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.

[10] M. Mitchell, J.P. Crutchfield, and P.T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

[11] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983.

[12] S. Wolfram. *Theory and Applications of Cellular Automata*. World Scientific, Singapore, 1986.