

# Using Genetic Algorithms to Evolve Behavior in Cellular Automata

Thomas Bäck<sup>1,2</sup>, Ron Breukelaar<sup>1,\*</sup>

<sup>1</sup> Universiteit Leiden, LIACS, P.O. Box 9512, 2300 RA Leiden, The Netherlands  
{baeck,rbreukel}@liacs.nl

<sup>2</sup> NuTech Solutions GmbH, Martin Schmeißer Weg 15, 44227 Dortmund, Germany  
{baeck}@nutechsolutions.de

**Abstract.** It is an unconventional computation approach to evolve solutions instead of calculating them. Although using evolutionary computation in computer science dates back to the 1960s, using an evolutionary approach to program other algorithms is not that well known. In this paper a genetic algorithm is used to evolve behavior in cellular automata. It shows how this approach works for different topologies and neighborhood shapes. Some different one dimensional neighborhood shapes are investigated with the genetic algorithm and yield surprisingly good results.

## 1 Introduction

Evolutionary Algorithms is the name for the algorithms in the field of Evolutionary Computation which is a subfield of Natural Computing and already exists more than 40 years. It was born 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 most widely known instances are genetic algorithms [9–11], genetic programming [12, 13], evolution strategies [16–19], and evolutionary programming [7, 6]. A detailed introduction to all these algorithms can be found e.g. in the Handbook of Evolutionary Computation [1].

Today the Evolutionary Computation field is very active. It involves 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 is based on a program loop that involves simplified implementations of the operators mutation, recombination, selection, and fitness evaluation on a set of candidate solutions (often called a

---

\* 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”*.

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.

No attempt will be made to give a complete introduction or overview of evolutionary algorithms, as there are many good introductory books on the topic available, e.g. [1], instead an example of how to use evolutionary algorithms to parameterize other algorithms will be given.

Evolving parameters for complex algorithms could also be viewed 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 inverse design of cellular automata (CA) is such a problem. 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 when it comes to real world problems. This paper uses a genetic algorithm to generate the transition rules for cellular automata, thus evolving global behavior with local rules using a genetic base.

## 2 Cellular Automata

According to [20] 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 can be viewed as 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  $C = \{a_1, a_2, \dots, a_N\}$  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  $N = 149$ . 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  $\forall n = 1, \dots, N$   $a_n^{t+1} = \Theta(s_n^t)$ . Given  $C^t = \{a_1^t, \dots, a_N^t\}$ ,  $C^{t+1}$  can be defined as  $\{\Theta(s_1^t), \dots, \Theta(s_N^t)\}$ .

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,  $\Theta$  can be written as a row of ones and zeros  $R = \{b_1, b_2, \dots, b_{2^{2r+1}}\}$  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 [14] 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 3). A genetic algorithm is used to evolve rules for one dimensional CA that do a good job of solving the Majority Problem [14] and it is shown how these rules seem to send ‘‘particles’’ and communicate by using these particles [15]. 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.

Previous work [4] suggested that using multi dimensional CA works a lot better than using one dimensional CA. This paper will try to shed light on what effect a different topology or neighborhood shape can have, and a measure will be given to classify these different CA.

### 3 Majority Problem

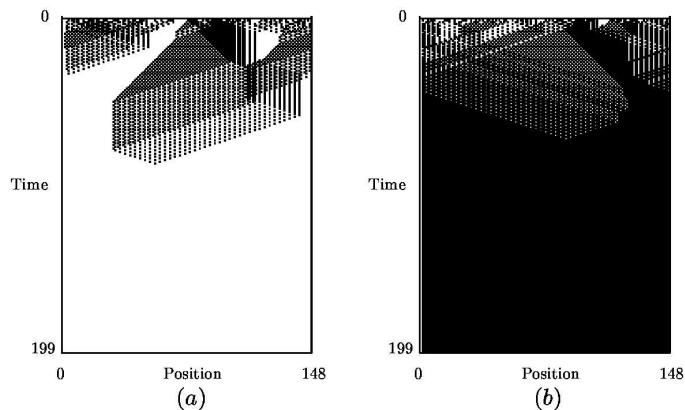
One of the best known global problems that is (partly) solvable with local rules is the Majority Problem. The Majority Problem can be defined as follows:

*Given a set  $A = \{a_1, \dots, a_n\}$  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 has to be solved within the framework of 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  $\lambda$ , 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*



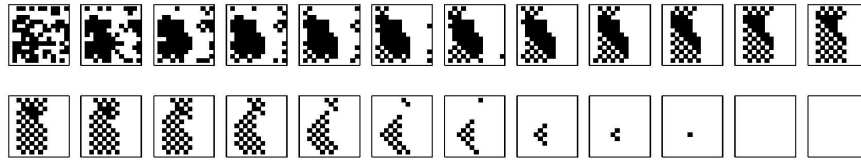
**Fig. 1.** These are examples of majority problem classification by the rule found by David, Forrest and Koza. [5]. Both are correct classifications (a) with 74 ones in the initial state, (b) with 75.

*every cell in the CA is zero and the ‘all one’ state being a the state in which every cell is one.*

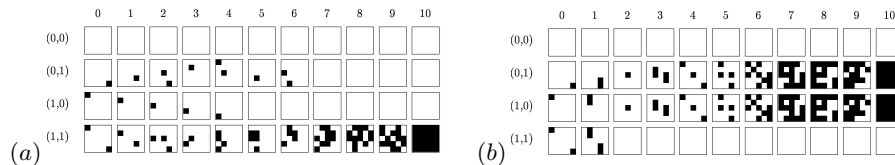
Evaluating a transition rule for this problem is done by iterating  $M$  randomly generated initial states and calculating the relative number of correct classification. The fitness of a transition rule is denoted with  $F_{N,M}$  where  $N$  is the width of the CA. The fitness can be calculated with different distributions over the number of ones in the initial state, but the default is a binomial distribution (denoted with  $F_{N,M}^B$ ) where every cell in the CA has a 50% chance of being initiated with a one for every initial state.

The first intuitive rule to come up with is 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. The majority rule gets stuck on the problem that on the boundary thick line in the time plot the cell 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 neighboring cell 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 fails.

Researchers in the field of cellular automata have published many different rules to solve this problem, one such rule is the GKL rule after Gacs, Kurdyumov and Levin [8]. This rule is pretty 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 L. Davis found a better one in 1995 which did 81.8%. In the same year R. Das found a rule that did 82.178%. Then in 1996 David,



**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  $F_{169,10^3} = 0.715$ .



**Fig. 3.** This figure shows two iteration runs of transition rules for two dimensional CA that were evolved using a GA. (a) shows how a CA can behave like an AND-port and (b) shows how it can behave like an XOR-port.

Forrest and Koza found a rule by cleverly using genetic programming that was able to classify 82.326% correctly [5].

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 percent of all random initial states correct, for this shows there is some kind of communication going on; some kind of emerging behavior.

## 4 Inverse Design

Nature has some remarkable examples of local rules that exhibit global behavior. Ant colonies are a good example. An individual ant does not seem to be very intelligent and does not seem to know what is doing and why, but a colony of ants seem very organized and purposeful. This did not happen over night, but evolved over millions of generations.

The global behavior of a CA can also be evolved in a similar way by evolving the local behavior. This “inverse design” of the behavior is done by using a genetic algorithm to evolve the transition rule that described the behavior. The fitness function of the genetic algorithm then defines the desired behavior.

M. Mitchell, J. P. Crutchfield and P. T. Hraber have shown [14, 15] that using a simple GA to evolve transition rules for the majority problem (explained in Section 3) can already give surprisingly good results. About half of the rules that were found performed better than the most trivial rule and about 7 rules out of

300 rules were found that seemed to use some primitive form of communication that worked for more than 70% of the classifications. This is not better than rules that are made by hand, but it does show how a GA can evolve global behavior based on local rules.

In [3] this idea was extended to two dimensional CA and it was shown that two dimensional CA can match the performance of the one dimensional rules even though the neighborhood was two cells smaller. It was also shown that different kinds of problems can be solved using this approach (AND-, XOR-problem and pattern generation) if it is extended to two dimensions. Figure 3 shows how a two dimensional CA can be evolved to behave like an AND or XOR port.

A generalization was defined in [2, 4] so that the approach could work for  $n$ -dimensional CA and different sizes of neighborhoods. The GA was altered to make it more robust and some experiments were done to compare results for different dimensionality's on identical problems. The way the transition rules were tested was also altered to more accurately describe the desired behavior. The results in [4] suggested that it is a lot easier to find rules for a three dimensional CA than it is for a one dimensional CA. Even though the number of cells in the neighborhoods are identical.

This article will more strongly support the claim that the topology of the CA and its neighborhood is very important for the performance of that CA. Four different neighborhood shapes are chosen to show this relation and to give indication in what shape of neighborhood might have the best performance and why.

## 5 Distance Measure

A normal one dimensional CA (as described in Section 2) has a very simple defined neighborhood, being "all the cells within a certain radius  $r$  of the center cell". Although this seems to be the most logical neighborhood CA can have many different shapes.

Every iteration step in a synchronous CA the state of every cell is updated using the information in the cells of the neighborhood of that cell. That means the way the information moves ("travels") through the CA is defined by the shape of the neighborhood. Note that because the standard one dimensional neighborhood is symmetrical, if information travels from cell  $a_i$  to cell  $a_j$  it will also travel from cell  $a_j$  to cell  $a_i$ . The number of iterations it takes for information to travel from cell  $a_i$  to cell  $a_j$  can be called the "distance between  $a_i$  and  $a_j$ ". If the shape of the neighborhood is different, the distance between cells in the CA will be changed too. Not only will this change the behavior of the CA, but it might also change the possible behaviors of the CA. That means that two neighborhood shapes might performance different on the same problem.

Intuitively, to solve a global problem with local rules information from every cell in the CA has to travel to every other cell in the neighborhood to be combined in a way that suits the problem. Because combining information results in

new information and the space in a CA is fixed, there will be information loss. Therefore combining the information as fast as possible seems to be good way to counter this information loss and solve the problem in the best possible way. This then would mean that the distance between two cells in the CA need to be minimized so that the information can be combined faster.

To measure the rate at which information is combined in a CA two metrics are defined. The “maximum distance between two cells” and the “average distance between two cells” in a CA. Because every cell has the same neighborhood shape every cell has the same maximum distance to a cell and the same average distance to all the cells in the CA. Note that these metrics are very dependent on the shape of the neighborhood and the size and topology of the CA.

If it is true that minimizing the distance in a CA will improve the performance of the CA then this might explain why multi dimensional CA seem to be more powerful than the one dimensional CA [3,2]. The topology of multi dimensional CA supports information travel in multiple directions and this decreases the distance between cells in the CA. An experiment was conducted to compare different neighborhood shapes in combination with the well known Majority Problem (see Section 3) and measure what the impact of the different distance measures is on the performance. In order to test the performance of a neighborhood shape a GA was used to search for a good transition rule that solves the Majority Problem like was done in [2–4, 14, 15].

## 6 The Genetic Algorithm

The GA in this paper is the same as used in [4]. This GA evolves a pool of 100 transition rules to find the one with the best performance. It uses tournament selection as defined in [1] to select which individuals to keep alive. This 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  $t + 1$ . This is repeated until generation  $t + 1$  has the same number of individuals as generation  $t$ . In this experiment  $q = 20$ .

After selection recombination is applied to generation  $t + 1$ . Recombination 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 used in the crossover is denoted by the crossover-rate  $c$ . Mutation is done by flipping every bit in the individual with a probability  $m$ . In this experiment  $c = 0.9$  and  $m = 2/l = 1/64 = 0.15625$ .

All the individuals in the pool are initialized at random with a normal distribution over the number of ones in an individuals bit string. This means that the number of individuals with a certain number of ones will be roughly equal to the number of individuals with any other number of ones. 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 best answer. Note that

Name	Physical distance	Max. distance	Avg. distance
Normal	-3 -2 -1 0 1 2 3	25	12.79
Exponents of 2	-4 -2 -1 0 1 2 3	19	9.97
Exponents of 3	-9 -3 -1 0 1 3 9	10	5.48
Exponents of 5	-25 -5 -1 0 1 5 25	6	3.84

**Table 1.** This table shows the physical layout of the four different neighborhoods used in this experiment. The maximum and average distance (as described in Section 5) are calculated on a CA with 149 cells.

this does not need to be the case if the fitness function used in the algorithm is probabilistic. In this experiment  $D = 100$ .

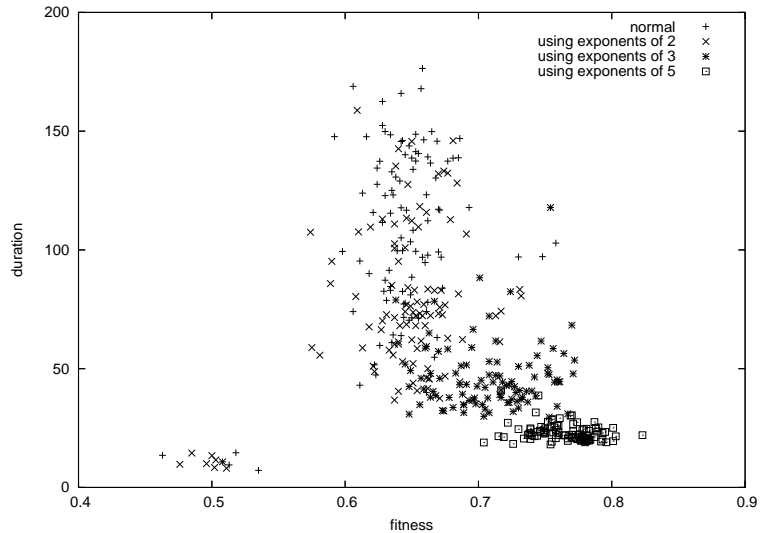
For the fitness evaluation a one dimensional cellular automata is used with a width of 149 cells. Every cell has 7 cells in the neighborhood, but the shape of this neighborhood will be different for every test. The transition rule will therefore consist of  $2^7 = 128$  bits and there will be  $2^{128} \approx 3.4 \cdot 10^{38}$  different rule each with a different behavior. Every generation 100 initial states will be generated that are used to calculate the fitness. The percentage that a rule classifies correctly of these 100 initial states is the fitness of that rule. A state is correctly classified if the iteration of the CA stops changing before 320 steps in an “all ones state” or an “all zeros state” corresponding to  $\lambda > 0.5$  and  $\lambda < 0.5$  (see Section 3).

## 7 Experimental Results

In this experiment four different neighborhoods will be tested. The first one will be the standard one dimensional neighborhood including cells 1, 2 and 3 on both sides of the center cell. The other three are chosen to minimize the distance between cells in the neighborhood and include the cells with a physical distance of an integer to the power of 0, 1 and 2. The three integers chosen for this are 2, 3 and 5. Table 1 gives an overview of the layout of the four neighborhoods.

For every neighborhood 100 runs are calculated with the GA as described in Section 6. Unsurprisingly the results of the normal neighborhood matched that of results in [4] and are only a little bit better than the results in [14, 15]. The rest of the results are a lot less trivial though. The neighborhood with “exponent of 2” is performing slightly better than the normal neighborhood, whereas “exponents of 3” performs a lot better with more then half of the rule topping 0.7 and about 10% over 0.75. And “exponents of 5” is even better than all the rest with all the rules found being above 0.7 and some even topping 0.8 with the best at 0.813 coming very close to the best rule found on Majority Problem. Furthermore the average number of iterations that a rule needs to classify an initial state gets smaller the higher you make the exponent. Figure 4 shows the best rule from all the runs.

These results support the claim that the shape of the neighborhood is very important for the performance of the CA and that decreasing the distance between cells in the CA increases the performance and decreases the duration.



**Fig. 4.** This figure shows all the rules found with the four different neighborhoods. Note how the fitness goes up and the duration goes down if the neighborhood is wider. The fitness is calculated using  $F_{149,10^3}$ .

Further research seems needed to determine whether this is the reason that multi dimensional CA have a better performance and a smaller duration [4], but the results suggest that changing the neighborhood or topology of the CA can change the behavior in a very drastic way. Taking this into account it does not seem very surprising that multi dimensional CA have a very different performance than one dimensional CA. It seems intuitive that the way information travels through a CA is an important factor in this and that therefore the metrics of distance (as described in Section 5) seem a very nice way to determine what neighborhood will have a high fitness and a low duration.

## 8 Conclusions

This article gives an example of how a genetic algorithm can be used to program an algorithm (in this case the cellular automata) and improve the results. Although there are probably better rules than the ones found in this experiment, the search space of transition rules for these cellular automata is so big that improving results is already a big achievement.

It also shows how using a GA it can be concluded that the shape of the neighborhood of a CA can have a big effect on the performance. Although the distance between the cells in a neighborhood is not proven to be the biggest factor in this, it does seem to be an indicator that can be used to devise the right shape of neighborhood.

## 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. Th. Bäck, Breukelaar R., and Willmes L. Inverse design of cellular automata by genetic algorithms: an unconventional programming paradigm. *UPP proceedings in the 'Hot Topics' subline of LNCS*, 2005.
3. R. Breukelaar and Th. Bäck. Evolving transition rules for multi dimensional cellular automata. In *6th International Conference on Cellular Automata for Research and Industry, ACRI*, Amsterdam, The Netherlands, 2004. Springer.
4. R. Breukelaar and Th. Bäck. Using a genetic algorithm to evolve behavior in multi dimensional cellular automata. In *(to be published) GECCO proceedings*, 2005.
5. 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.
6. D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, New York, 1995.
7. L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons, 1966.
8. P. Gacs, G. L. Kurdyumov, and L. A. Levin. One dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informatsii*, 1978.
9. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
10. D. E. Goldberg. *The Design of Invocation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.
11. J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
12. J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, 1992.
13. 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.
14. 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.
15. M. Mitchell, J.P. Crutchfield, and P.T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.
16. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
17. I. Rechenberg. *Evolutionsstrategie '94*. Fromman-Holzboog Verlag, Stuttgart, 1994.
18. H. P. Schwefel. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. *Interdisciplinary Systems Research*, 26, 1977.
19. H. P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.
20. S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983.