

Problem Solving by Evolution: One of Nature's Unconventional Programming Paradigms

Thomas Bäck^{1,2}, Ron Breukelaar¹, Lars Willmes²

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 evolution. The goal of the cellular automata designed by the evolutionary algorithm is a bitmap design problem, and the evolutionary algorithm indeed discovers rules for the CA which solve this problem 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 [7, 8], evolution strategies [11–14], 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 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 CAs 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 CAs evolution process. In the next section, we will explain the concept of a CA in some more detail. Section 3 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 [15] 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 a width N . 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 , then every a_n with $1 \leq n \leq N$ is connected to its left and right neighbors.

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 $2r + 1$.

There are different kinds of methods to change the values of a CA. The values can be altered all at the same time (synchronous) or at different times (asynchronous). Only synchronous CA were considered for this research. In the synchronous approach at time step t each value in the CA is recalculated according to the values of the neighborhood using a certain transition rule $\Theta : \{0, 1\}^{2r+1} \rightarrow \{0, 1\}, s_i \rightarrow \Theta(a_i)$. This rule can be viewed as a one-on-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 whole CA C at time t as C^t so that C^0 is the initial state (*IC*) and $a_n^{t+1} = \Theta(s_n^t)$. This means that C^{t+1} can be found by calculating a_n^{t+1} using Θ for all $1 \leq n \leq N$. Given $C^t = \{a_1^t, \dots, a_N^t\}$, C^{t+1} can be defined as $\{\Theta(a_1^t), \dots, \Theta(a_N^t)\}$.

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 $R = \{b_1, b_2, \dots, b_{2^{2r+1}}\}$ 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 $3,4 \cdot 10^{28}$) each with a different behavior.

The two dimensional CA used in this paper do not differ much from the one dimensional CA discussed so far. Instead of a row of positions, C now consist 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 can be defined in the same rows of bits (R) as defined in the one dimensional case. For a 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, this also means that searching in that space might take more time and finding anything might be a lot more difficult.

This research was inspired by earlier work in which transition rules for one dimensional CA were evolved to solve the Majority Problem [9,10]. The genetic algorithm used here is a fairly simple algorithm with a binary representa-

tions of the rules, mutation by bit inversion, proportional selection, and without crossover.

In the next section experimental results on the bitmap problem are reported.

3 Using Evolutionary Algorithms to Program Cellular Automata

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. To make the problem harder and to stay in line with earlier experiments the CA has unconnected borders. To make the problem even more challenging the von Neumann neighborhood was chosen instead of the Moore neighborhood and therefore the s_n consist of 5 cells ($r = 1$) and a rule can be described with $2^5 = 32$ bits. The search space therefore is $2^{32} = 4294967296$. 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 0 except the position ($\lfloor \text{width}/2 \rfloor, \lfloor \text{height}/2 \rfloor$) which is 1. The theory being that this ‘seed’ should evolve or grow into the desired state in the same way as a single seed in nature can grow into a flower.

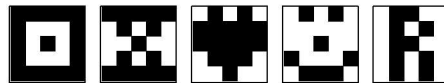


Fig. 1. The bitmaps used in the pattern generation experiment.

For this experiment only mutation was applied as an evolutionary operator. Mutation is performed by flipping every bit in the rule with a probability P_m . In this experiment $P_m = 1/\{\text{number of bits in a rule}\} = 1/32 = 0.03125$.

In trying to be as diverse as possible five totally different bitmaps were chosen, they are shown in figure 1. 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 1 shows the number of successful rules for every bitmap. Note that symmetrical bitmaps seem to be easier to generate than asymmetric ones.

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. Figure 2 shows the behavior a few successful transition rules generated by the GA (in each row, the evolution of the CA is shown from left to right). Note that different transition rules can end up in the same desired state and have totally different iteration paths.

Ongoing experiments with larger CAs suggest that they do not 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 the size of the CA increases.

Table 1. Number of successful rules found per bitmap.

<i>Bitmap</i>	<i>Successful rules (out of a 100)</i>
“square”	80
“hourglass”	77
“heart”	35
“smiley”	7
“letter”	9

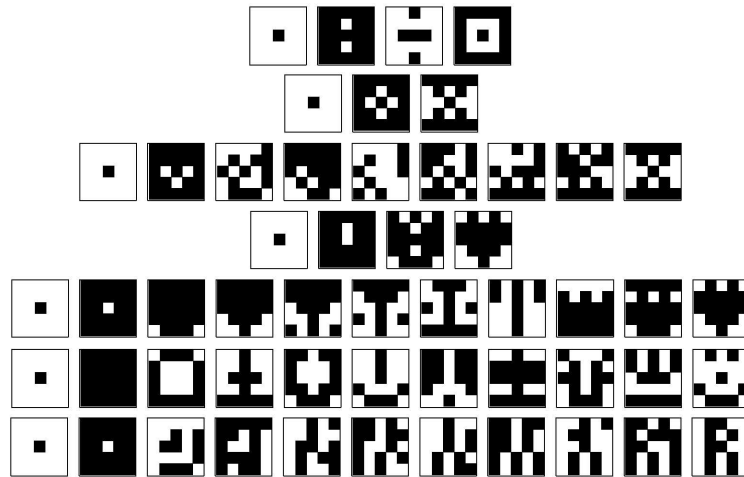


Fig. 2. This figure shows some iteration paths of successful transition rules.

4 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 CAs is possible by means of evolutionary computation in a clear, straightforward, and very powerful way. The results clearly indicate that real world

applications of CAs could also be tackled by this approach, and the unconventional programming of CAs by means of EAs 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. J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA, 1992.
8. 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.
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. I. Rechenberg. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.
12. I. Rechenberg. *Evolutionstrategie '94*. Fromman-Holzboog Verlag, Stuttgart, 1994.
13. H. P. Schwefel. Numerische optimierung von computer-modellen mittels der evolutionstrategie. *Interdisciplinary Systems Research*, 26, 1977.
14. H. P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.
15. S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983.