Yelizaveta Gorstko
CBSSS, Caltech
Summer, 2002

Learning with Noise: Cellular Automata approach.

In this paper we give a brief overview of Cellular Automata (CA's) and their use in computation. We discuss potential applications of stochastic CA's and possible changes in the operators used in Genetic Algorithms that could make evolving stochastic CA's more efficient.

Yelizaveta Gorstko
CBSSS, Caltech
Summer, 2002

Introduction: Biology, Computation, and Hardware

Today attention of many scientists turns more and more often away from the sequential Von Neumann architecture towards "nonstandard" ways of computing. The main reason and motivation for this lies in the fact that too many systems around perform computations in a quite disparate way. For example, the brain, human society, and insect colonies can perform complex tasks with no "central controller" to ensure that each individual component does what it is "supposed" to do. An increasingly interesting question is to find out how these completely decentralized systems, which components act in parallel, accomplish a common goal. Inherent parallelism and highly local interactions among cells make a Cellular Automaton (CA) a natural object for attempts to carry out computation in a way similar to the one that takes place in the brain.

This paper looks at a CA as a medium for computation and asks how it could be possible to obtain a CA capable of solving a particular problem. Since it is unrealistic to manually describe the correct behavior of such a CA, we do it via a Genetic Algorithm (GA). There exists quite a big number of works devoted to evolving CA's with a GA; however, all of them consider a software implementation of the GA. A different direction is hardware implementation of a GA. The step from software to hardware requires more work than one might initially suspect. Apart from building physical circuits and gates, engineers have to deal with a big real world problem: unreliability of physical components. In the software-GA, if a cell reports that it is in a particular state, for example '0,' then it is definitely in that state, but hardware, especially with a tendency towards diminishing sizes, can have components that are less reliable. As a result a cell that has state '0' might be regarded as having the state '1' by some of its neighbors. Evolving CA's in such "noisy" conditions should differ from evolving them in deterministic environment. Our approach to modeling evolution of a CA in the presence of noise is to use a stochastic CA. Nondeterminism in the behavior allows for new definitions of genetic operators used in the GA. These operators could potentially make the algorithm more effective than if it simply disregarded the noise altogether and proceeded as if in an ideal environment. Here we will discuss several possible changes to genetic operators.

Another interesting question deals with a comparison of evolution of deterministic versus stochastic CA's. Of course, the search space for the CA with nondeterministic behavior is larger than that of deterministic one; on the other hand, we are not trying to find particular probabilities, in a sense the final goal is still a deterministic CA, but if genetic operators that manipulate probabilities instead of definite states appear more effective than the usual ones, they may yield better and faster results.

This paper is devoted to a discussion of questions and possible approaches; deciding which ones can produce good results requires practical implementations. The author of this paper implemented a short program (in Lisp) to test the newly defined genetic operators (discussed in the section on GA's). Unfortunately, one step of a GA involves multiple (about 300) runs of CA on different instances of the problem (density task in our case), and each run of CA on a regular desktop computer takes (in our implementation) approximately 45 seconds. This number implies that one run of a GA takes on the order of two weeks, and considering that fine-tuning the GA requires finding good values for a number of parameters used by the genetic operators, the realistic time

for coming up with a good algorithm is at least several months. However, this does not impede a discussion of possible ideas for future work, which could, for example, avoid the time bottleneck by using the Cellular Automata Machine (CAM) designed by Prof. Margolus and his team in the 1980s in order to speed up running of CA's by creating special architecture that used parallelism and local and uniform interconnections.

Here we will shortly discuss CA's in general and stochastic CA's in particular, then mention the main ideas behind a GA and discuss our ideas about possible changes in the genetic operators that could yield better results in a noisy environment, as well as possibly in a regular environment.

Cellular Automata

An n-dimensional Cellular Automaton consists of an n-dimensional array of cells and a finite set of states. Each cell carries one state (which in some cases may be a word that also carries information about the past state) and knows the states of several cells in its r-neighborhood, for example with r=3, each cell knows about itself, three cells to its right, and three cells to its left. In one time step all cells simultaneously update their states depending on the states of the cells in their r-neighborhood (Note: there are other ways of updating, for example "Margolus neighborhood" partitions a CA into disjoint blocks, and these blocks, not cells, are updated simultaneously; partitions alter in time, so that new blocks overlap with the old ones. This definition, useful for constructing simulations of reversible systems, is not needed in our case.).
Although, in this paper we look at CA's as a medium for computation, they are often used for simulations of physical systems. Many regard CA's as discrete counterparts to differential equations: they allow description of continuous systems with a finite number of cells that have a finite number of states and follow predefined rules. The widespread interest in CA's first arose due to their ability to simulate complex dynamics by simple local interactions.

Once it has been shown that CA's are capable of Universal computation, which means that potentially one could find a CA that was equivalent to any finite algorithm, their computational ability started to attract interest as well. CA's are usually distinguished by their behavior: some are too simple to be capable of attacking any interesting task, some are too unpredictable, for example chaotic: little differences in the initial distribution of states produces large differences in the limiting behavior, but there are some that have a potential to form interactions necessary for computation. In particular many of the later works deal with nonuniform CA's, where rules differ for cells in different positions; the possibility to have different behaviors in different parts of a CA seems to allow more freedom in defining more interesting behaviors.

Above we have described a deterministic CA, i.e. once a cell knows the neighborhood configuration (the set that consists of the states of cells in the cell's r-neighborhood), it can take one and only one state. On the other hand, a stochastic CA has cells that follow some probability distribution during the update: once a cell receives the information about the neighborhood configuration it finds this configuration in its rule table, makes a random choice between 0 and 1 and follows the corresponding probability distribution to decide which state to take based on this distribution and the random choice. For example, in a two-state CA ('0' and '1') a configuration (0 1 0 0 1 0 0) with

a corresponding probability distribution (‘0’:0.3 ‘1’:0.7) a random choice of 0.4 would make the cell change its state from 0 to 1. Stochastic CA’s have been used for various problems, such as simulations of stochastic processes, behavior of imperfect (“noisy”) systems, and prediction of a long-term behavior of deterministic systems.

In our project we did not deal with imposing “noise” on top of a deterministic CA and looking if it could be possible to still achieve expected behavior. It was noticed that CA’s that are capable of computation are incredibly “brittle,” i.e. introducing noise breaks completely their ability to perform computation. That’s why, from the very beginning we wanted to look not at the final, ready-to-use, CA, but at the means of obtaining a stochastic CA. Once such a CA is obtained, a detailed analysis of its rule table can provide information about what level of noise it can stand. The early works that deal with stochastic CA’s looked for the optimal distribution of probabilities; however, their goal was not to find a CA that is maximally robust in the presence of noise. An algorithm that introduces a penalty for rules that involve extremes, for example that have the probability of changing to ‘0’ or ‘1’ go too near to 1, can direct the CA towards being more nondeterministic. Once we have the final CA, we can look at the rule tables for each cell and analyze how well this CA can perform given the level of our background noise. The final CA can be regarded as a deterministic CA (the state corresponding to the biggest probability in the rule gives the deterministic state: this is equivalent to “majority rules” approach of running a nondeterministic CA many times, or running multiple copies of it) and the variation (probabilities of choosing other states) give the maximum interference, i.e. background noise, that could be imposed on the CA. This analysis can show several things: first, whether or not this CA would be functional at all in the imposed conditions; second, if reducing noise is costly, in which parts we can allow having more noise, as well as which parts of the CA are the most sensitive to noise and have to be protected. These possible uses of a stochastic CA serve as a strong enough motivation for looking at the ways to obtain one capable of a particular computation task.

Genetic Algorithms and Cellular Automata

GA’s get the solution (or a close approximation of it) to the problem not explicitly, by deriving it, but implicitly, through an evolutionary process. In a GA the well-known principle of the “survival of the fittest” applies not to live organisms, but to solutions of the problem. GA consists of the following main components: a fitness function that, based on the results of an individual’s performance subject to one or several tests, evaluates how close each individual comes to satisfying the requirements that the true solution should satisfy, a selection mechanism that picks out individuals for reproduction based on their fitness, and genetic operators of crossover and mutation that have to shift the properties of offspring individuals closer to satisfying these predefined requirements. In our case solution is the CA. However, dealing with multiple CA’s requires too much time. If we want to apply a GA to find this solution, we should consider rules in each cell as individuals. A natural choice for a “gene” is the probability distribution that corresponds to a particular neighborhood configuration; a “chromosome” should contain distributions corresponding to all of the configurations, i.e. the whole rule. Thus one individual here is encoded with one chromosome. We start with a population of

randomly generated rules and evolve them for about a hundred generations. The above description already pinpoints the main advantage and disadvantage of a GA. We do not need to explicitly know the structure of the solution to be able to obtain it. This makes GA's a good choice for the problems where it is hard to figure out the precise relations and dependencies, but easy to see whether or not a potential solution satisfies our requirements. On the other hand, it is often hard to derive the fitness of an individual based on its genotype, which means that it is necessary to subject it to tests and evaluate fitness based on its phenotype. More tests given to each individual make the fitness estimate better. However, running these tests may take a long time. Indeed, tests used in fitness evaluation take the biggest part of the entire GA, and this part tends to become a serious bottleneck in the case of many complex problems. A potential solution is a hardware implementation of a GA, which allows time reduction by several orders of magnitude. And, as we have mentioned before, knowledge about evolving stochastic CA's will most probably be useful for hardware implementation.

Detailed descriptions of a GA used to evolve a deterministic CA designed for a density task are given in the works by Mitchell and Sipper. Similar to their works, we have a one-dimensional CA with 149 cells designed to perform the density task: decide whether a given list of 149 0's and 1's has more '0' or not. Length of the neighborhood, r, is 1, i.e. each cell considers a list of three elements: states of its immediate left and right neighbors, and its own, as a configuration that allows updating the current state. The given problem provides initial states for the cells, and these, i.e. '0' and '1,' are all the states that we use.

A side note: here the number for the length of the problem equals the number of cells. Biological systems often amplify the signal before producing the reaction. The existing models all show increasingly poor behavior for problems where the number of 0's approaches the number of 1's. This range of poor behavior gives a threshold on sensitivity to differences in the quantity of 0's and 1's for a CA that does not amplify the input signal (problem). If a CA replicates the problem several (k) times, the difference gets multiplied by k also, thus decreasing the range of poor performance. One way that CA's could achieve better performance might be by amplifying signals and increasing the length. Of course, if more cells make it more costly, it will have to balance performance with length. This is again an optimization problem, which suggests another interesting application of a GA: to find k that gives the preset error range given the penalty for the wrong behavior and a penalty for extra cells.

In order to evaluate fitness, rules have to be subjected to tests. Here is one of the interesting characteristics of CA's: it is impossible to assess each individual rule completely on its own. The whole idea is that individual components <u>together</u> lead to some meaningful behavior, and we don't know what each individual component is supposed to be doing. Behavior of each individual is too closely interwoven with behavior of the cells in its r-neighborhood. The only way to assign fitness to a rule is to run the entire CA on "many" problems, each of which has a known solution, and see if the given cell reaches the correct "limiting" state in each case. The ratio of the number of correct final states to the total number of tests gives a good approximation of the fitness of a cell. Of course, since the simulation has to end in a reasonable time both "many" and "limiting" have to be given finite values that are both sufficient and realistic. The numbers used here are 300 problems for "many" and 320 steps for "limiting."

The problems are randomly generated. The number of problems in the "less 0's" and "less 1's" categories are also picked out randomly from a uniform distribution. This ensures that a cell that simply always sticks to one of the choices, i.e. exhibits the "1/2 behavior," does not appear as a good one to the genetic algorithm. It is highly improbable that a CA with performance of 1 (a 100 per cent correct) exists; it definitely does not among uniform, one dimensional CA's. The rule derived manually by Gacs, Kurdyumov, and Levin in 1978 for a uniform CA with r equal to three, defined by the "majority vote" among: if current state is '0': you, left neighbor, cell 3 sites to the left, and if current state is '1': you, right neighbor, cell 3 sites to the right, performs as well as 0.98, and the CA's obtained via GA's get as high as 0.95-0.96. This shows that expected behavior of the stochastic solution should definitely be much higher than 0.5 and should even be approaching 0.9's.

The selection mechanism is responsible for both pushing the evolving population in the desired direction and making sure that it does not exhibit the "fast convergence" phenomenon, which happens when the population converges too quickly to an individual that dominates the rest of population, but does not satisfy the requirements well enough. The dominance of its genes may delay or prevent better solutions from appearing during one run of a GA, i.e. in a hundred generations. In the works by Mitchell, fast convergence is mentioned as one of the biggest problems. They always choose the best individuals for reproduction; however, this selection impedes transitions to new "stages" (the current stage changes with evolvement of a radically new behavior). For example, CA that has discovered the one-half behavior, under a fitness function that does not punish this behavior well enough, may have a few cells that have this behavior, and if the other cells perform worse, the selection mechanism may quickly force all the cells to be their copies, thus eliminating diversity in the population and taking away the whole meaning of a genetic algorithm, which is not designed to obtain all the diversity in the gene pool exclusively from the effect of genetic operators, in particular mutation. In order to avoid this problem, we have implemented the "roulette wheel" selection, where the probability of being picked out are in direct proportion with fitness, and yet the best individuals are not guaranteed a chance to contribute to the creation of the next generation.

Finally, the genetic operators, which receive the set of parents from the selection function and output the set of offspring, were the main point of interest of our project. In all of the works mutation played a tiny role and all the emphasis was given to the crossover operator. The probability of mutation was a small constant, independent of the fitness of an individual. We decided to explore the role of mutation in more detail. Sipper insists on introducing locality from the very first stages: the parent cells for an offspring come only from its neighborhood, which is similar to what happens in real life. Initial assignment of rules is random, so on the first stages the cell finds itself evolving with a random, but heavily reduced gene-pool. Neighborhoods overlap, so eventually changes do propagate, but, maybe, for nonuniform CA's it could be more efficient also to employ mutation to solve the problem of gene reduction. After all, the species on initial stages of the development of life on the Earth were prone to mutations. Apart from individual fitness values, it might be useful to define for each cell the fitness of its neighborhood (for example, just an average of fitness values of all cells in its r-neighborhood). Mutation operator can take the neighborhood fitness and, based on it, adjust the probability of mutating (changing a random gene of) the rule. Instead of a constant line at some level,

the mutation operator with probability as a function of the neighborhood fitness, looking like a concave function (through the (0,1) point) that sharply declines to this constant line, could help balance gene diversity and reproduction locality for the CA. The role of the crossover operator, which swaps pieces of two chromosomes at the random breaking point, is to direct the population towards the higher overall fitness. An interesting problem here arises due to the ordering of the genes in the chromosome. Success of the crossover operator depends on whether or not it can preserve structures in the neighborhood relations that are important for the computation. Of course, we don't know which exactly dependencies to preserve. The standard way to arrange the genes in the chromosome (rule) is by the ordering the neighborhood configurations lexicographically, as words. Since this ordering is imposed by us and most probably is not of any significance to the CA, this choice is equivalent to picking a random order once and keeping it for the rest of the evolutionary process. It is possible that varying this order could bring better results. Instead of choosing a random point and switching the pieces of two chromosomes, we pick a random number for the total number of genes involved in the crossover, as a proportion of the total length of the rule and randomly pick out this number of genes from the rule. Varied random crossover instead of the fixed one may give better fit offspring in the long run. The next question deals with what we should do, once we have compiled these two lists of genes that are involved in the crossover operation. The standard method is simply to swap them. However, we are dealing with a stochastic CA, moreover with a possible plan to impose the noise conditions on it. It might be useful to consider whether or not the two genes (note: not the two original chromosomes) that interact during crossover have an equal say in this process. Suppose we know that one cell is definitely more reliable than the other, where reliability means an accurate report about its own state. Suppose further that it is possible to test cells before the GA starts, and we know that one cell makes a mistake with a frequency of 0.3 and another - with a frequency of 0.7. Unlike deterministic CA's, stochastic CA's allow for an arithmetic manipulation of the probabilities in each gene, which suggests the following possible crossover operation: take the probability distributions of both genes and output the weighted (by 0.7 and 0.3 - the probabilities of correct outputs) average of these probability distribution. It might also be useful to make weights depend on the neighborhood fitness as well. The weighted averages may help to take into account both the reliability of cells and their relative fitness. Biological crossover involves simple switching of genes. Yet nature created such notions as dominant and recessive genes, and these ideas are absent in algorithms that work with a single-stranded chromosome. There are some works devoted to "dominant crossover" in genetic programming; however, typical GA's do not implement any form of dominance outside of the selection mechanism. Weighted averaging may be one of the possible ways to accomplish this for stochastic CA's.

Conclusion

We have discussed the possible uses of stochastic CA's and ideas that could help efficient implementation of GA's that could evolve them. All of the suggestions are not hard for implementing in any language that handles lists well, and with an access to a

CAM machine, which could eliminate the time constraint, are quite realistic for practical experiments. These experiments would show which of the ideas are indeed beneficial and which only complicate the algorithm without making it at all more efficient. We hope that, apart from giving a short background and overview of CA's and GA's, this discussion can lead to some serious attempts of evolving and studying stochastic CA's.

References

Lee et. al.

Adaptive Stochastic Cellular Automata: Theory

Adaptive Stochastic Cellular Automata: Applications

(Physica D: Nonlinear Phenomena. Cellular Automata: Theory and Experiment, 1990)

Mitchell, M. et. al. :

Revisiting the edge of Chaos: Evolving Cellular Automata to Perform Computations

Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments

Evolving Cellular Automata with Genetic Algorithms: A review of Recent Work

Sipper, M. :

Co-evolving Non-Uniform Cellular Automata to Perform Computations

Co-evolving architectures for cellular machines

Green, Kirley :

Connectivity and Catastrophe - towards a general theory of evolution

Investigation of a Cellular Genetic Algorithm that Mimics Landscape Ecology

General:

Gacs*, Reliable Cellular Automata with Self-organization*

J. von Neumann*, Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*

Acknowledgments

This paper is a result of the ***"Computing Beyond Silicon" Summer School*** held at the *California Institute of Technology* in 2002. Without the lectures and discussions during the CBSSS the author would not have had a chance to find out and, as a result, think about computation with Cellular Automata.

The author would like to thank the organizers of the CBSSS program:
Dr. Andre DeHon and **Dr. Erik Winfree** whose support and criticism incited and directed the investigation about stochastic CA's and GA's.
And also **Dr. Margolus** whose lectures and discussions showed CA's at work, as models of physical systems, and encouraged the interest in the object that has such strictly local interconnections and yet is capable of such unison and coordination.