Evolving Cellular Automata
NCASO Final Project

Eric Severson
eseverson@ucdavis.edu

UC Davis Department of Mathematics

Abstract

We revisit work by Crutchfield, Mitchell, et al. (1993-1998) on using a genetic algorithm to evolve one-dimensional cellular automata rules that solve the Majority Classification and Synchronization tasks. The genetic algorithm solutions to Majority Classification were dominated by asymmetric block-expanding strategies. For Synchronization, we found perfect solutions which used domain and particle interactions to synchronize the whole lattice. Finally, we explored the effects of increasing alphabet size, and found improved block expanding strategies for Majority Classification and a more efficient solution to the Synchronization task.

1. Introduction

1.1. Why Cellular Automata?

Despite the spatially-localized, deterministic nature of cellular automata, these processes can exhibit complex, global emergent behavior. By searching for CAs that performed computational tasks, the previous papers explored both how spatially local systems can perform global computation, and how evolutionary processes can lead to such solutions.

For the purposes of this project, CAs are a rich dynamical system that produce interesting and often unpredictable behavior. Through exploration of this work in evolving task-solving cellular automata, we hope to gain better understanding of emergent global patterns in CA and potentially discover new types of CA behavior.

1.2. CA Framework

In this work, we are considering one-dimensional cellular automata acting on a lattice of N cells over a k-state alphabet. Each cell has a neighborhood which includes itself and the r closest left and right neighbors, for some fixed radius parameter r. The lattice has periodic boundary conditions, so the cells on one edge neighbor the cells on the other. A CA is defined by its update rule, which maps all possible neighborhoods to the states a cell in that neighborhood will update to. The CA acts upon the lattice by simultaneously updating all cells via the update rule applied to their neighborhoods. An example CA rule is below, for alphabet $k = 2$ and radius $r = 1$:

Neighborhood	111	110	101	100	011	010	001	000
New State	1	0	0	1	0	1	0	1

Table 1: CA rule 10010101

With the neighborhoods listed in lexicographic order, the CA rule can be represented as a string of length k^{2r+1} (the number of possible neighborhoods). These rule strings are how the CAs are represented in our genetic algorithm.

Another way to label the CA rules is to consider this string as a number in base-k, so our above is example is also described as ECA rule 149 (the 256 binary-alphabet, radius-1 rules are the Elementary Cellular Automata).

Figure 1 shows the space-time diagram for this example rule. A random initial configuration of length 50 is updated by this rule for 20 time steps. The horizontal slices in this figure are the one-dimensional lattice configurations

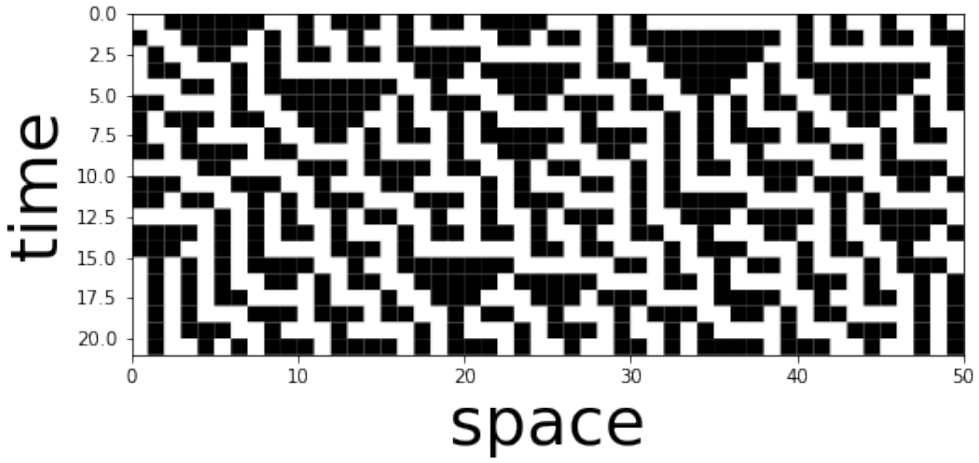


Figure 1: space-time diagram for the example rule 10010101

at each time step, where time is running from top to bottom. The letters 0 and 1 in the binary alphabet appear as white and black squares, respectively. These diagrams are quick ways to visually intuit the behavior of a given CA rule, and thus will be the main way we show the behavior of our CAs in this paper.

2. CA Tasks

In our work below, we will consider two different tasks for the cellular automata:

2.1. Majority Classification

The first is the Majority Classification task. Genetic algorithm solutions to Majority Classification were the topic of most of the reference literature ([1],[2],[3],[4],[8], and [10]). The task can be described as follows:

Given an arbitrary initial configuration over a two-state alphabet, the CA must converge to a stable configuration of all 0 or all 1, indicating which symbol comprised the majority of the initial configuration. To make the majority always well-defined, we will only consider initial configurations with an odd lattice size N.

The performance score, P_N^I of a CA rule on this task is the proportion of I random initial configurations which are successfully classified by the CA within a limit of M time steps (here we use $M \approx 2N$ to give sufficient time for the CA to propagate information through the lattice).

The difficulty in this task lies in the need to compute a global quality of the initial state, when the CA rule can only use local interactions. To demonstrate these difficulties, consider a naive candidate solution for this problem: the Local Majority rule. This rule will update each cell with the state that represents the local majority among states in its neighborhood.

Figure 2 shows this Local Majority rule acting upon a random initial configuration. Within the first couple of time steps, the lattice is updated to local blocks of all 0s and all 1s, but this local block configuration is a steady-state, because the white and black cells along the boundaries each have a majority of white and black cells in their respective neighborhoods. This naive solution is unable to resolve these locally homogeneous regions, so will fail to classify most initial configurations.

A better performing hand-designed rule is the GKL rule, designed by Gacs, Kurdyumov, and Levin [12]. It was not constructed for the purpose of this task, but is able to solve most initial configurations ($P_{149}^{10^4} \approx 81\%$). The GKL rule is a binary alphabet, radius 3 CA whose update rule can be described as follows: if a cell is in state 0, replace it by the majority between itself and the cells one and three units to the left. Similarly, if a cell is in state 1, replace it by the majority between itself and the cells one and three units to the right.

Figure 3 shows the GKL rule acting on two different configurations. Initially, the configuration converges locally to regions of all 0 and all 1, similar to the Local Majority rule. There is a similar static boundary between white all-0

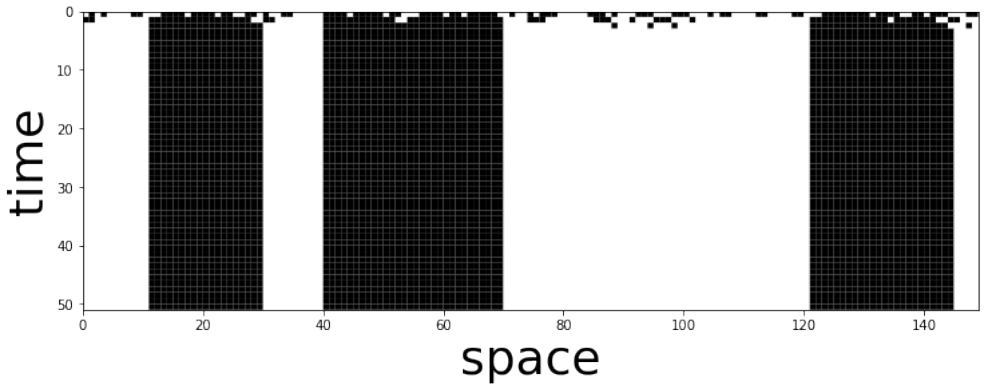
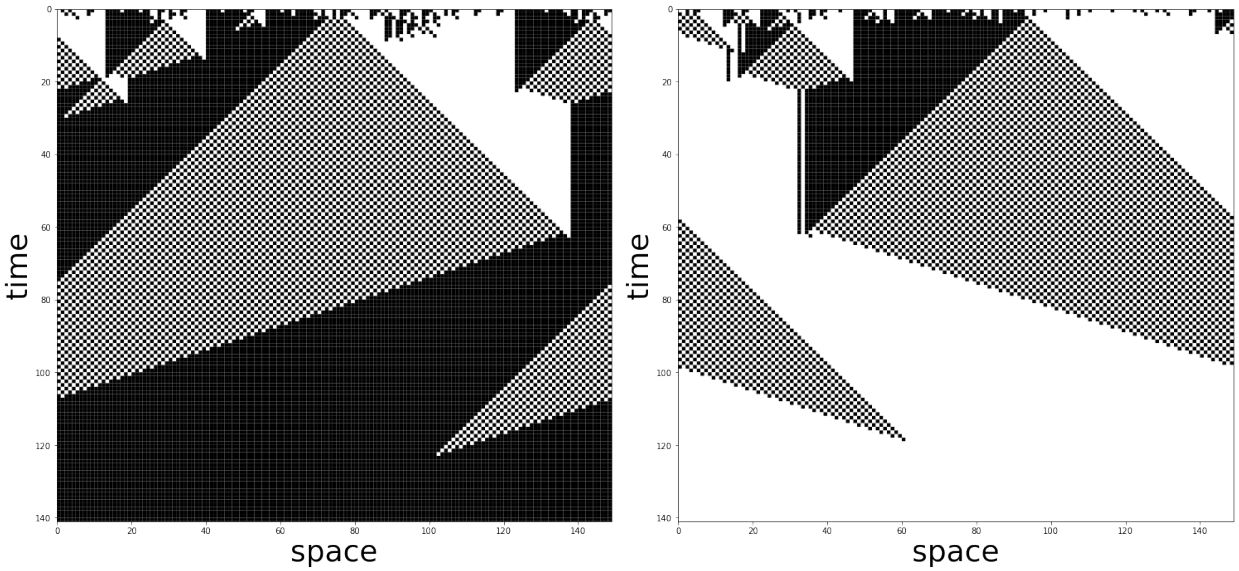


Figure 2: space-time diagram for the Local Majority rule with radius 3, acting on Initial Configuration A.



(a) Initial Configuration A (55%1s)

(b) Initial Configuration B (45%1s)

Figure 3: The GKL rule correctly classifying two different Initial Configurations.

regions on the left and black all-1 regions on the right. However, when an all black region on the left borders an all white region on the right, a new checkerboard pattern is propagated out in both directions. When the checkerboard pattern reaches the boundary of either region, it starts contracting at a faster speed. The result is that the smaller region is annihilated and the larger region wins out. In in the center of Figure 3a we see the checkboard region annihilate the white region to the right, so the all black pattern eventually takes over, whereas the opposite result happens in the center of Figure 3b, with the black region annihilated.

The all-white, all-black, and checkboard patterns are all examples of domains: spatially homogeneous regions which are mapped to themselves by the CA rule. The GKL rule's computational behavior can thus be understood by the interactions between the different domains and the moving particles which are defined by the different domain boundaries. For more in-depth domain-particle analysis of this rule, see [3].

As mentioned above, the GKL rule successfully classifies about 81% of random initial configurations. It is worth noting that the density of 1s in a random initial configuration will be binomially clustered around 0.5, and these almost-tied configurations are the hardest to classify. Data from [1] suggests that the GKL rule only fails to classify configurations whose densities are within some distance of 0.5, and this distance shrinks as the lattice size N increases.

It has also been proven that no small-radius, binary CA rule can perfectly solve majority classification [11]. The best rules known to date for the $r = 3$ majority classification can classify about 89% of random initial states, and were found using more sophisticated evolutionary techniques [13].

2.2. Synchronization

The Synchronization task was first considered in [6] and analyzed further in [8] and [9].

Given an arbitrary initial condition, the CA must reach a state of global synchronous oscillation between the all-0 and all-1 configurations. Here the final state is not reflecting any information about the initial state. The difficulty in this task is resolving local phase discrepancies. Consider a naive solution which maps all neighborhoods to 0, except for all the 1 neighborhood, which must be mapped to 0 to create oscillation behavior. With $r = 1$, this would correspond to the rule 00000001 (or ECA rule 1).

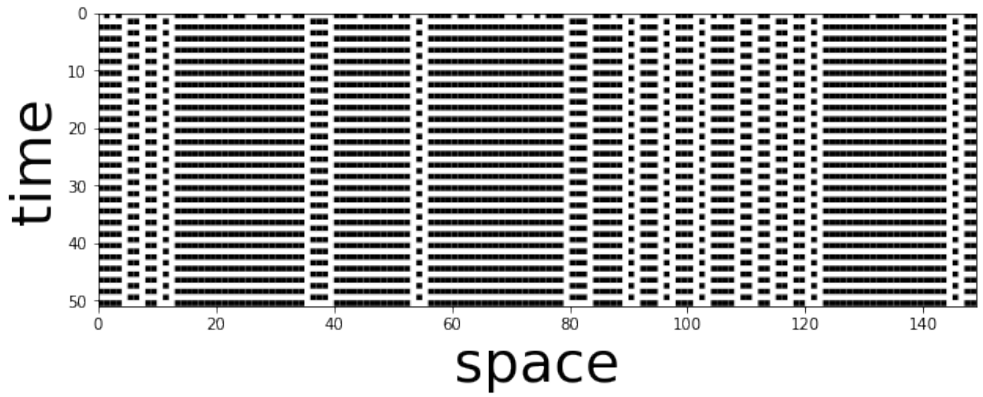


Figure 4: space-time diagram for rule 0000001, acting on IC A, which fails to achieve global synchronization.

Figure 4 shows the behavior of this rule. We see that large portions of the lattice become synchronized, but some small regions are out of phase, and this phase discrepancy is never resolved.

Unlike Majority Classification, the Synchronization task can be solved on 100% of random initial configurations. Perfect solutions using $r = 3$ were found in both the literature [6] and in our own Genetic Algorithm runs.

3. Genetic Algorithm

The number of CA rules is $k^{k^{2r+1}}$. Beyond Elementary CAs, the size of the rule space is too large for an exhaustive search. For the binary $r = 3$ CA setting, the rule space is 2^{128}. To try to search for effective solutions in such a large

space, the papers utilized a genetic algorithm. A description of our GA, modeled after that of the reference papers, is given below:

The genetic algorithm starts with initial population p of random rule strings. Instead of each rule having an independently random bit for each letter in the rule string, the initial population is chosen so the distribution of the density λ of 1s in the rule strings is uniformly distributed over [0, 1]. This was found to lead to more effective solutions by the GA in [3]. Our data agreed with these results, as trials with independently random bits did not evolve effective strategies, while those with uniform random initial rule density were more successful.

Each generation of the algorithm proceeds as follows:

First the population rules are evaluated according to a fitness function. A set of I initial configurations of length N are generated, and each rule is evaluated on these configurations. The fitness score F_I is the proportion that reach the desired state after $2N$ time steps (so either are classified correctly in Majority Classification or globally oscillate in Synchronization).

The set of initial configurations also has density of 1s uniformly distributed over [0, 1]. This leads to more high and low density cases, which are easier to classify or synchronize from. Testing on unbiased random initial configurations is a stricter metric of a rule's performance, but leads to worse results by the GA. Including more easy cases in the fitness test helps the early generations evolve.

The rules are then sorted by their fitness scores. The top proportion E of rules are copied directly into the next generation's population. The remainder of the population is formed by choosing two of these elite rules to cross over. A random bit l in the rule string is chosen, and the crossover rule has the first l bits from one rule and the rest from the other. This new rule is then mutated, where each letter in the rule string is replaced by a random letter with probability m (this lets the algorithm generalize to larger alphabet sizes).

This procedure is repeated for G generations, to find high fitness rules which perform the task effectively.

For our GA runs, we used parameters $p = 100$, $E = 0.2$, $I = 100$, $N = 149$, $m = 0.032$, $G = 50$, which were chosen to be match those used in the literature [10].

4. Majority Classification GA Trials

Our runs of the GA to perform Majority Classification were dominated by block-expanding strategies, which achieve high fitness despite being qualitatively different from the domain-particle computation of the GKL rule. Block-expanding strategies were the dominant evolved rule in 13 of our 15 trials.

The other 2 trials were stuck at a fitness of 0.51, with the simple early strategies which turned all neighborhoods to 0 (or 1), except for the neighborhood or all 1 (or 0). This strategies essentially always guessed one color, and only correctly classified the configurations with that color, or the single extreme configuration of all the other color.

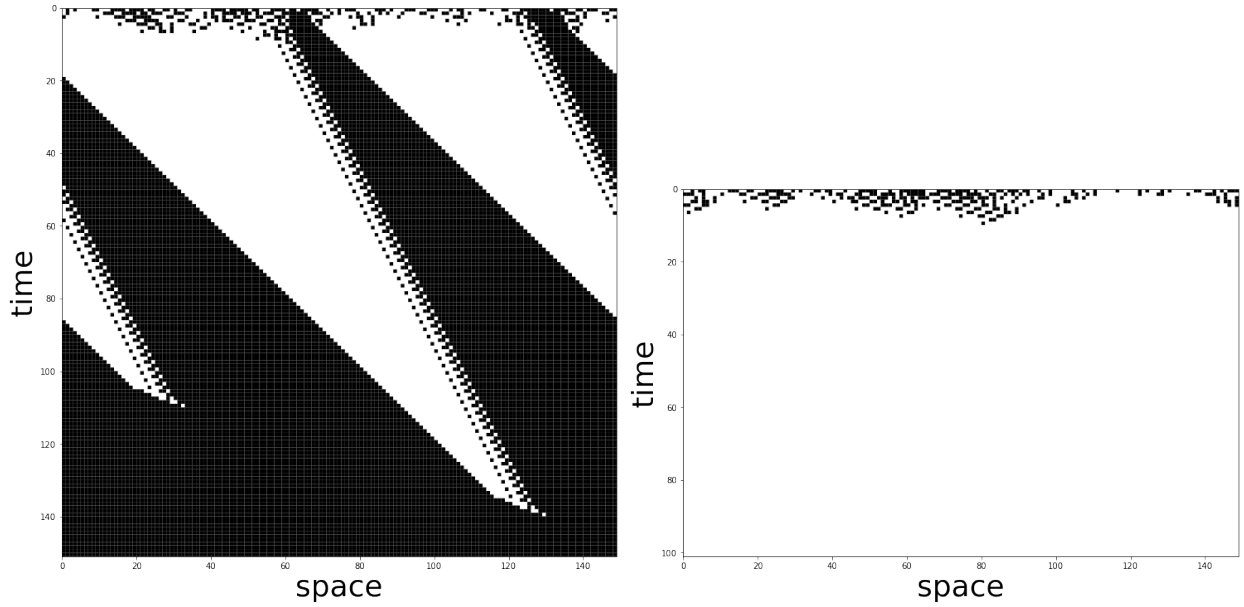
These results were similar to those in [10], where 280 out of 300 trials evolved block-expanding strategies, and 11 of 300 trials were stuck with the simple default strategies.

Figure 5 shows a black-block-expanding strategy where most regions converge to all white, but sufficiently large blocks of black are expanded and eventually fill the whole lattice. In 5a, the initial configuration has two black blocks which are expanded, whereas in 5b, the entire lattice becomes white.

Likewise, figure 6 shows a white-block-expanding strategy, a color-reversed version of the same idea. In all these examples, the block-expanding rules are correctly classifying the Initial Configurations A and B.

These block-expanding strategies are asymmetric, unlike the GKL rule and other highest-performing classification rules. Their success relies on the correlation between high densities of one letter and the probability of having a large block of that letter. They are also tuned to the lattice size N which they were evolved on. For larger lattice sizes, block expanding strategies get progressively worse. It can be proven that $P_N \to 0.5$ as $N \to \infty$, where P_N is the probability of a block-expanding strategy succeeding on a random configuration of size N. As lattice size increases, the probability of large blocks of a given letter will approach 1, so a block expanding strategy will classify almost all initial states to its block color, and thus only succeed on the 50% of configurations where that was the true majority. Both of these examples reached fitness scores above 0.9, but only classify about 68% of random initial configurations.

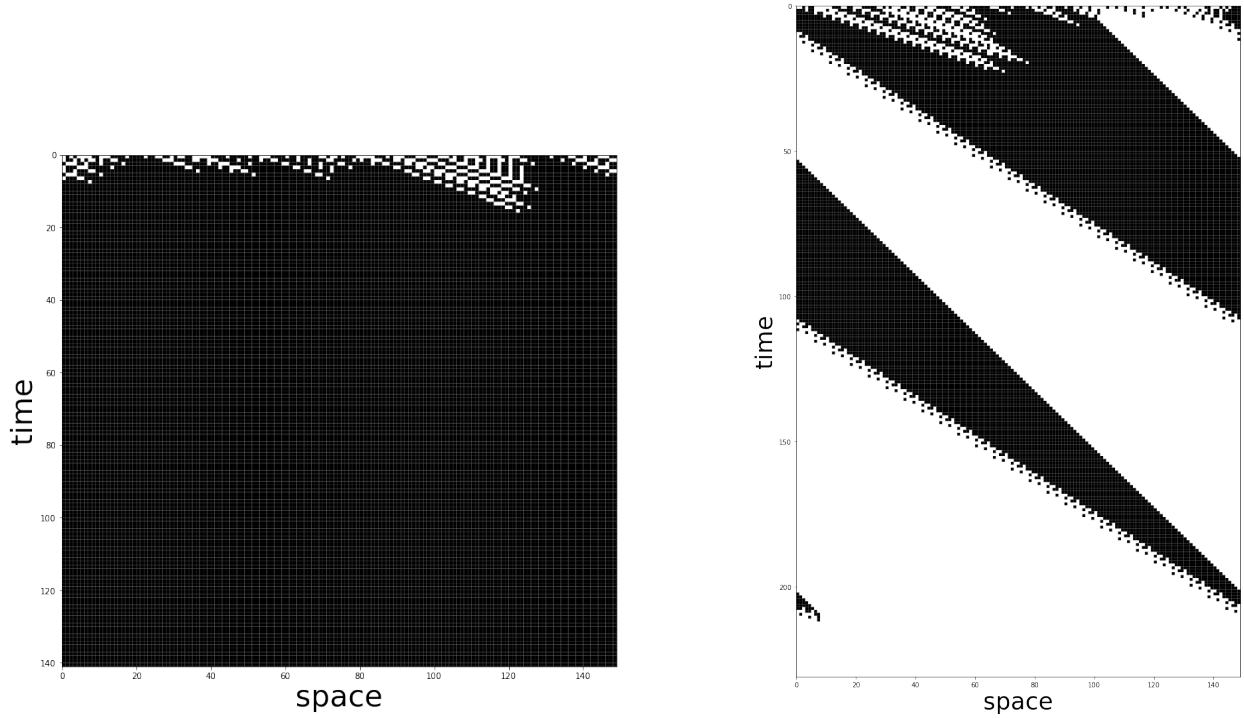
As we saw above, the GKL rule had a higher success rate of 81% on random $N = 149$ initial configurations and still succeeds for higher N. The reasons why these GAs failed to consistently evolve better particle-based methods are discussed more at the end of [3]. These early works did succeed in evolving some particle-based rules similar to the



(a) Initial Configuration A (55%1s)

(b) Initial Configuration B (45%1s)

Figure 5: A GA-evolved rule which expands black blocks.



(a) Initial Configuration A (55%1s)

(b) Initial Configuration B (45%1s)

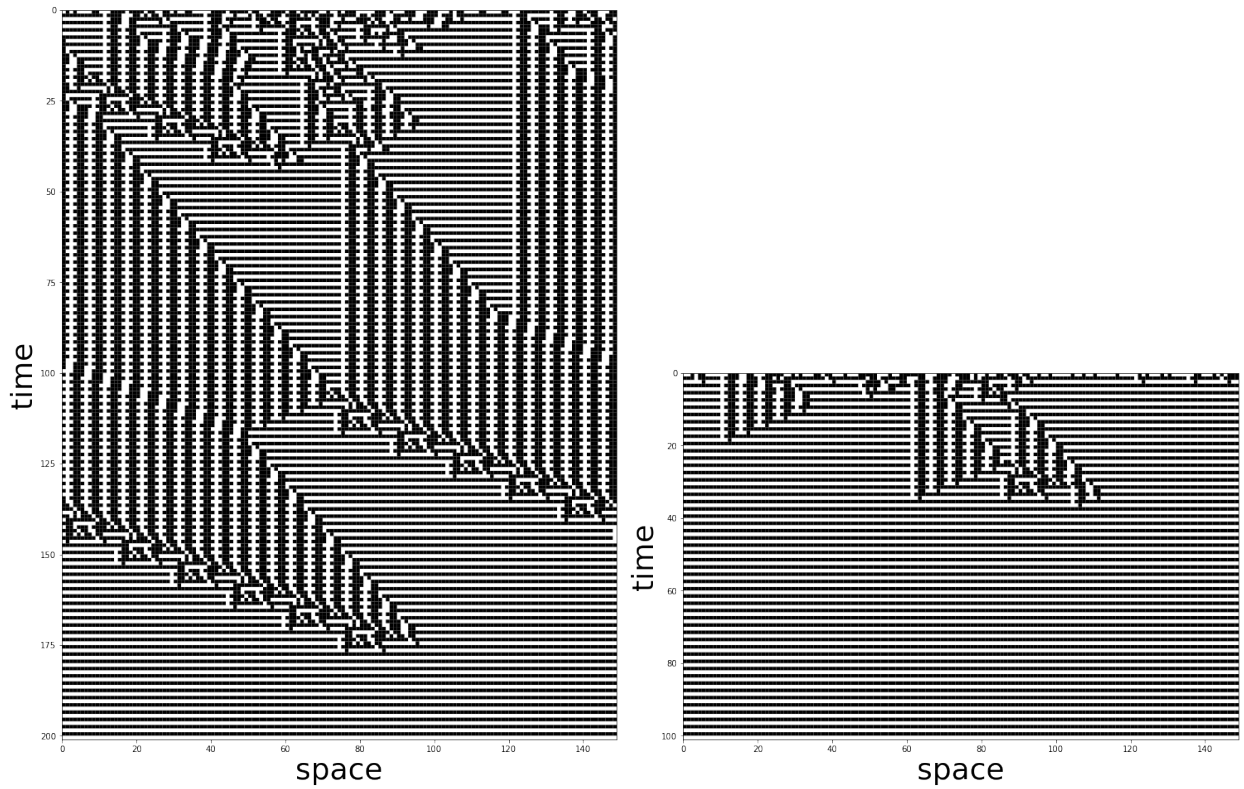
Figure 6: A GA-evolved rule which expands white blocks.

GKL rule, but they only found them in 9 of 300 runs [10]. Thus it was not surprising, although disappointing, to have only found block expanding rules, given our smaller number of trials.

Later papers improved the design of the evolutionary algorithm to consistently find higher performing particle-based strategies and avoid the local optima of asymmetric block-expanding strategies. The most effective rules to date were generated through a multi-tier evolutionary environment, with separate population groups and additional fitness criterion that evaluated internal symmetries which should be present in the most effective rule strings. [13].

5. Synchronization GA Trials

Our GA trials using the Synchronizaton task as the fitness function were able to find radius $r = 3$ rules which were able to synchronize 100% of initial configurations. An example rule is shown in Figure 7. It uses domain-particle interactions to synchronize the entire lattice, similar to the solutions from [6].



(a) Initial Configuration A

(b) Initial Configuration B

Figure 7: A GA-evolved rule with perfect scores on synchronization task.

The domain-particle interactions are made more clear by the visual analysis in Figure 8. There are two types of domain: S, which oscillates between (0)* and (1)*, and D, which oscillates between (11110)* and (10011)*. The * denotes a periodic repetition of this word. The interactions between different domains depend on the relative phase of each domain. D is created at the boundary between S and $\bar{S}$, where the bar denotes a domain in the opposite phase. The different particles defined by the boundaries of an S and a D domain are described below:

The interactions between multiple D domains depends on the relative phases and periods. The effect of all of these particle behaviors is that the D domain grows if it between two S regions of opposite phase and it contracts if it is between two S regions of the same phase. Thus our CA rule's domain-based strategy can be described as follows: the D domain is created between regions in opposite phase, and then expands until at least one of the regions

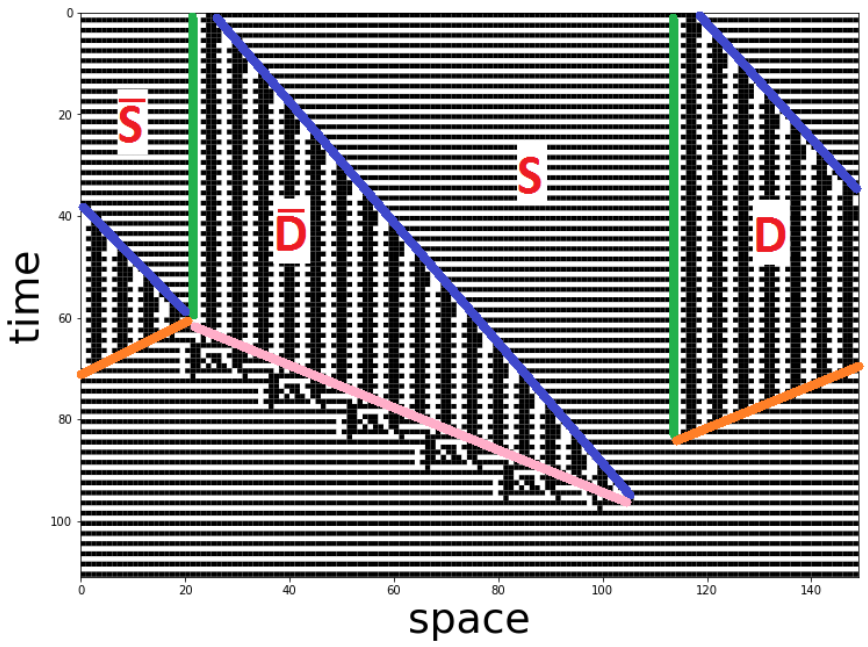


Figure 8: Visual analysis of the two domain types and the different boundaries between them. * denotes a region in opposite phase.

Domain Boundary	Color in Figure 8	Spatial Period	Velocity
$S\,D$ or $\bar{S}\,\bar{D}$	Green	2	0
$\bar{D}S$ or $D\bar{S}$	Blue	6	$\frac{5}{6}$
DS or $\bar{D}\bar{S}$	Orange	2	$-\frac{5}{2}$
$S\,\bar{D}$ or $\bar{S}\,D$	Pink	6	$\frac{5}{2}$

Table 2: Information on the common particles.

is annihilated. The D domain will then contract when it is between regions of the same phase, so eventually the entire lattice is left synchronized.

This type of domain-particle analysis was conducted more rigorously in [8] and [9]. Those works used an automated filter which recognized the regular domain languages, and then produced images of just the particle background. These particle interactions were also replicated in idealized models, in order to test the claim that they were the source of the computation. In our example, we were able to identify these properties through careful visual analysis, but a more in-depth automated approach will be useful for future works when we produce more complicated CA behavior.

6. Extending to Alphabet Size 3

The above results were all similar to what we found in the early literature. After finding these similar results in $k = 2$, $r = 3$ solutions to the Majority Classification and Synchronization tasks, we decided to explore the effect of increased alphabet size.

Each task is defined on a two-letter alphabet. Rather than trying to generalize the task to a larger alphabet size, we kept the task definitions and initial configurations defined using a binary alphabet. By increasing the CA's alphabet size, we are essentially giving the rule a third intermediate state it can use in the computation, even though the input and desired output configurations will only be comprised of the first two letters.

Because increasing the alphabet size greatly increases the size of the rule space, we reduced the radius to compensate for this increase in complexity. We hoped that the increased alphabet would compensate for the smaller radius.

The only further changes necessary to generalize our Genetic Algorithm to incorporate a larger alphabet was in the selection of the initial population of rules. Again, an unbiased random choice of initial rules, using a random letter for each bit in the rule string of each rule, led to very poor GA performance, as the early generations were not able to progress in their fitness scores. The next attempt was to keep the initial population using only 0 and 1 in their rule strings, with density of 1s uniformly distributed in the initial population rules, as was done in the previous trials. This way, 2 bits in the rule strings were only able to arise through mutations. This led to better results by the GA, but we were worried that this could be biasing solutions away from heavily incorporating the third letter into their patterns.

We ultimately had the densities of rule bits for our initial population rule strings be drawn uniformly at random from the k-dimensional simplex, generalizing the uniform 0-1 distributed we had for the $k = 2$ cases. This did not lead to noticeably different solution behavior compared to the previous method, however.

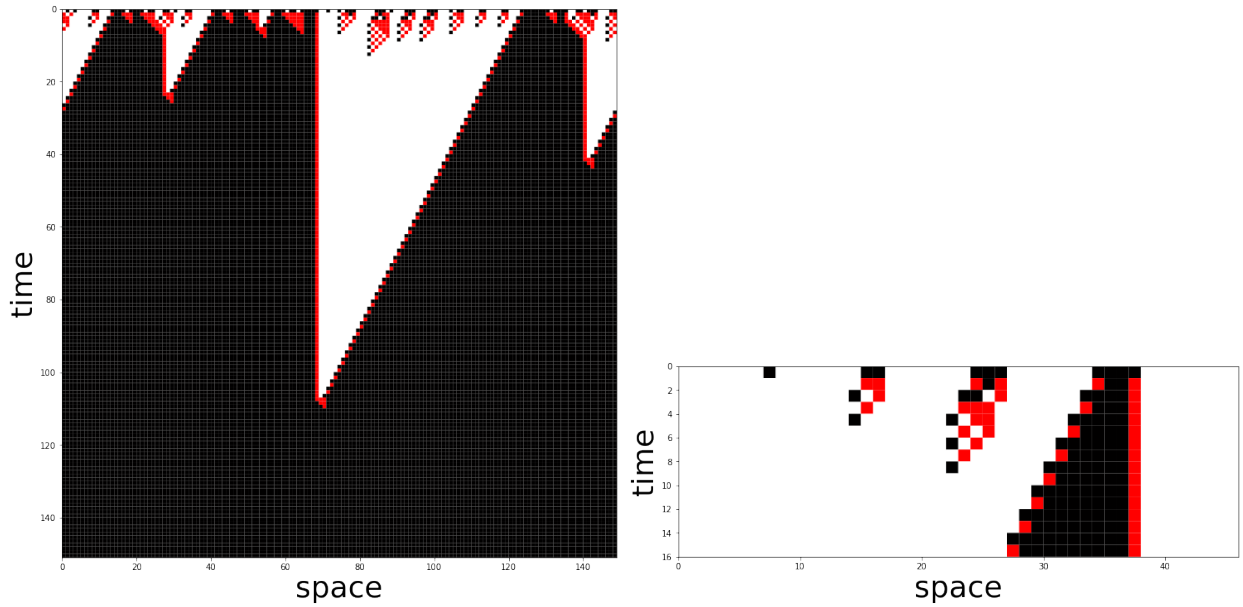
6.1. Majority Classification, $r = 1$

The first problem to consider was Majority Classification with radius 1 over this 3 letter alphabet. Without the third letter, we are in the class of Elementary Cellular Automata, none of which can classify more than 50% of random initial configurations.

Thinking about the increased alphabet size, we predicted finding an improved block expanding strategy. A hand-designed rule we found was able to expand only blocks of size 4 or greater. This level of tolerance was not possible without the larger alphabet.

The highest fitness rules found by the GA exhibited exactly this type of strategy. An example black-block-expanding rule is shown in Figure 9. In 9b, we see how this rule uses the third letter, here appearing as red, to restrict the size of blocks which get expanded. Red letters appear at the boundaries of black blocks. In the case of block of size 3, these two first red blocks are both within the neighborhood of the center cell, which then turns white, effectively stopping this block from future expansion. For a block of size 4, however, both red edges are not within the same neighborhood, so this block is able to expand indefinitely.

This block-expanding rule was able to achieve fitness scores of ≈ 0.78, better than the best ECA results of 0.51. However, it still only succeeds on 50% of random initial configurations with $N = 149$, because this block size threshold of 4 is too small.



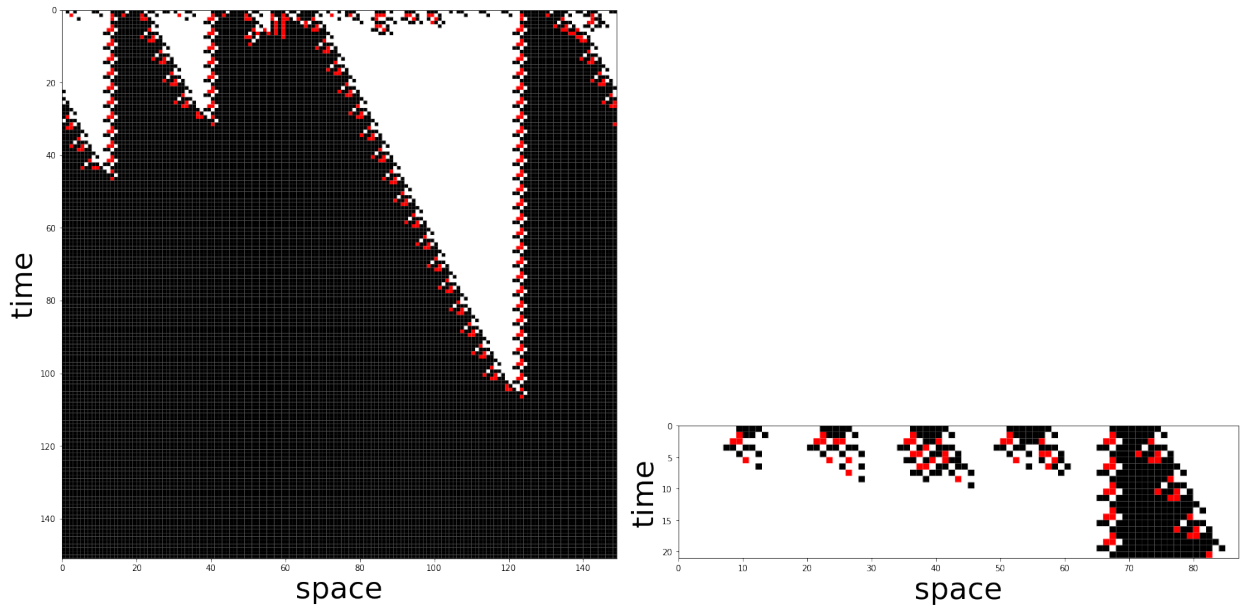
(a) Initial Configuration A (55% 1s)

(b) Black blocks must be size 4 to be expanded.

Figure 9: An alphabet 3 radius 1 rule that exhibits improved block-expanding, found by the GA.

6.2. *Majority Classification,* $r = 2$

Our GA trials with alphabet 3 radius 2 were qualitatively similar. Figure 10 shows one of the highest fitness $r = 2$ rules that was found in the trials, which expands sufficiently large black blocks.



(a) Initial Configuration A (55% 1s)

(b) Black blocks must be size 8 to be expanded.

Figure 10: An alphabet 3 radius 2 rule that exhibits improved block-expanding, found by the GA.

This top performing rule had fitness scores of ≈ 0.93 and correctly classified $\approx 65\%$ of random initial configurations. These numbers were comparable to the best performing block-expanding strategies that our GA had evolved with $k = 2$, $r = 3$. This suggests that the increased alphabet size is able to make up for the reduced radius.

As another comparison, trials with $k = 2$, $r = 3$ had top fitness scores of ≈ 0.85 and classified $\approx 54\%$ of random configurations. Again increasing the alphabet size is noticeably improving fitness.

6.3. *Synchronization,* $r = 1$

Finally, we considered the Synchronization task with alphabet 3, radius 1. Similarly to our previous Synchronization trials with $k = 2$, $r = 3$, we found rules with perfect fitness scores which were able to synchronize on 100% of random initial conditions. However, the solutions were qualitatively different, and did not rely on domain-particle interactions to propagate information globally through the lattice. An example rule is shown in Figure 11:

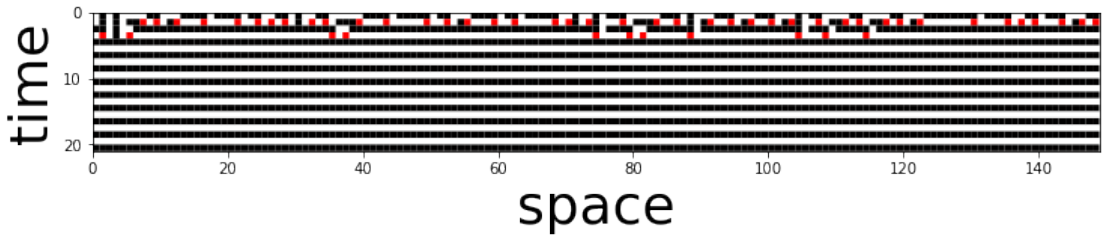


Figure 11: An alphabet 3, radius 1, CA is able to very quickly synchronize the whole lattice.

Here we see the CA rule is able to synchronize the entire lattice within only 4 time steps. The strategy used by the CA is made more clear in Figure 12. It is always able to synchronize such that the initial configuration is the black

part of the phase. It does this by using the third letter to create a boundary between regions that are out of phase. This letter only appears on the time steps when the correct phase is the 0 or white state. The result is that the correct phase always expands, while the out of phase regions always contract. Figure 12 shows an initial configuration with only three small regions of 1s. The synchronized regions around these small black islands expand to quickly synchronize the whole lattice such that the initial configuration was in the all 1 phase.

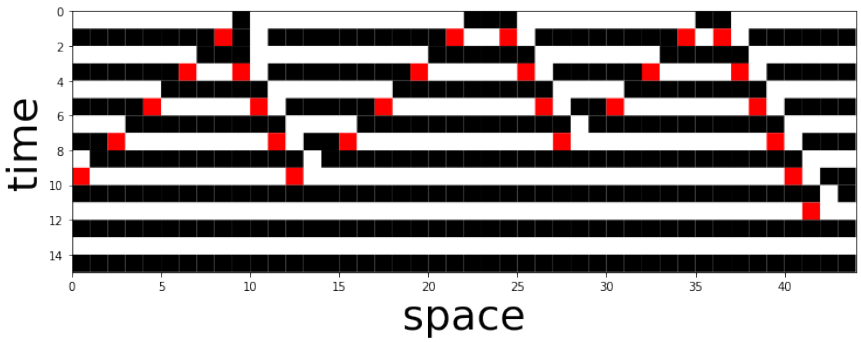


Figure 12: This rule always synchronizes to the black cells in the initial configuration.

In this example, we found that the increased alphabet size enabled a fundamentally new type of strategy. The third letter is essential to preserving the black phase in the initial configuration and allowing the lattice to so quickly synchronize.

7. Conclusion

We constructed a genetic algorithm similar to that described in previous works such as [10], and used it to search for solutions to the Majority Classification and Synchronization problems. The evolved strategies we found in the binary alphabet, radius 3 regime were similar to those reported in that literature: dominance of block-expanding strategies for Majority Classification and particle-based perfect solutions to synchronization.

Our trials did not, however, lead to any particle-based solutions for Majority Classification. Given that these were found over a much larger sample of trial runs [10], one future step is to rebuild the CA simulator and GA code to run faster and help generate larger data sets. Our code for this project was written in Python, and rewriting it in a faster language should help this end. Also we plan to set our code up to run in parallel to help get more trial data.

Regardless of computational power, the methods in our genetic algorithm were simplistic compared to the more focused evolutionary searches in recent literature [13]. We plan to improve our genetic algorithm for future work. Hopefully the improvements that have been made toward tasks such as the $k = 2$, $r = 3$ Majority Classification can help our algorithm more efficiently search through the rule space of a higher alphabet. Our preliminary results showed that there is potential for increased alphabet size to enable novel solutions to these tasks, so this seems to be an area worth future exploration.

Another future direction is moving to 2-dimensional CAs. This extension to the Majority and Synchronization problems has also been considered in later work [14], [15]. Beyond these tasks, we are considering novel tasks where the CA has to evolve around a fixed landscape in the initial configuration. This could be represented as an extra letter, which is forced to be static by the update rule. This framework would allow us to define new tasks, such as connecting randomly placed nodes in two-dimensional space (inspired by the Traveling Salesman Problem.)

It is clear that in this project we have only scratched the surface of the work that can be done in exploring the variety of behaviors that can be found in the CA rule space. Future work will build off of the ideas explored in this project.

8. References

[1] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. *Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations*. Complex Systems 7 (1993) 89-130.

[2] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. *Dynamics, Computation, and the 'Edge of Chaos': A Re-Examination*. In Complexity: Metaphors, Models, and Reality, G. A. Cowan, D. Pines, and D. Meltzer (eds.), Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume 19, Addison-Wesley (1994) 497-513.

[3] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. *Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments*. Physica D 75 (1994) 361-391.

[4] James P. Crutchfield and Melanie Mitchell. *The Evolution of Emergent Computation*. Proceedings of the National Academy of Sciences, USA 92:23 (1995) 10742-10746.

[5] Rajarshi Das, Melanie Mitchell, and James P. Crutchfield. *A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata*. In Parallel Problem Solving from Nature-III, Y. Davidor, H.-P. Schwefel, and R. Mnner (eds.), Springer-Verlag (1994) 344-353.

[6] Rajarshi Das, James P. Crutchfield, Melanie Mitchell, and James E. Hanson. *Evolving Globally Synchronized Cellular Automata*. In Proceedings of the Sixth International Conference on Genetic Algorithms, L. J. Eshelman (ed.), Morgan Kaufmann (1995) 336-343.

[7] Melanie Mitchell, James P. Crutchfield, and Rajarshi Das. *Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work*. In Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96), Russian Academy of Sciences (1996).

[8] Wim Hordijk, James P. Crutchfield, and Melanie Mitchell. *Embedded-Particle Computation in Evolved Cellular Automata*. In Physics and Computation '96 (Pre-proceedings), T. Toffoli, M Biafore, and J. Le£o (eds.), New England Complex Systems Institute, (1996) 153-158.

[9] Wim Hordijk, James P. Crutchfield, and Melanie Mitchell. *Mechanisms of Emergent Computation in Cellular Automata*. In Parallel Problem Solving from Nature-V, A. E. Eiben, T. Bck, M. Schoenauer, and H.-P. Schwefel (eds.), Springer-Verlag (1998) 613-622.

[10] James P. Crutchfield. Melanie Mitchell, and Rajarshi Das. *The Evolutionary Design of Collective Computation in Cellular Automata*. Machine Learning Journal, submitted.

[11] Mark Land, Richard Belew. *No perfect two-state cellular automata for density classification exists*. Physical Review Letters. 74 (25): 15481550. (1995).

[12] P. Gacs, G. L. Kurdyumov, and L. A. Levin. *One-dimensional uniform arrays that wash out finite islands* Probl. Peredachi. Inform., 14:92-98. (1978).

[13] Wolz, D. and de Oliveira, P.P.B. *Very effective evolutionary techniques for searching cellular automata rule spaces* Journal of Cellular Automata 3 (2008), 289-312.

[14] Oliveira, Gina MB, et al. *Some Investigations About Synchronization and Density Classification Tasks in One-dimensional and Two-dimensional Cellular Automata Rule Spaces* Electronic Notes in Theoretical Computer Science 252 (2009): 121-142.

[15] Cenek, Martin. *Information Processing in Two-Dimensional Cellular Automata* Diss. Portland State University, 2011.