# Project 2: Computation in cellular automata

CS 523, Spring 2015

Colby Guiterrez-Kraybill
University of New Mexico
colby@kraybill.com

Whit Schonbein
University of New Mexico
71whit@cs.unm.edu

## ABSTRACT

A cellular automaton (CA) is an $N$-dimensional lattice, where each location in the lattice can be in one of $k$ possible states. At each time step, the state of each location is updated according to a rule that maps a surrounding neighborhood to a new state for that location. Despite their simplicity, CAs can solve non-trivial problems. In this paper we investigate the capacity of 1-dimensional CAs to solve the '$\rho = \frac{1}{2}$' problem, i.e., the problem of taking an initial binary string and iteratively updating its states so that the final outcome correctly classifies the density of `1`s in the initial string: If the string has more `1`s than `0`s, the final lattice should be all `1`s, and should be all `0`s otherwise. A genetic algorithm with mutation and crossover is used to search for solutions. We show that (i) CA that have access to more information regarding the string they are classifying perform better than those with less information despite having a larger space of possible solutions, and (ii) well-performing CA exhibit 'mutational robustness' in the sense that the CA continues to perform well despite random mutations. Finally, we modify the genetic algorithm to utilize a dynamic fitness function that gradually encourages solutions to harder initial strings and discourages reliance on previously found solutions; We show that this 'biased fitness' approach leads to better-preforming solutions in comparison to the unbiased version.

## Author Contributions

Colby Guiterrez-Kraybill wrote the code and did analysis. Whit Schonbein wrote copy, wrangled LaTeX, and assisted in analysis. Both authors worked together on understanding the project, designing experiments, and interpreting the data.

## 1. INTRODUCTION

A cellular automaton (CA) is an $N$-dimensional lattice, where each location in the lattice can be in one of $k$ possible states. At each time step, the state of each location is updated according to a rule that maps a surrounding neighborhood to a new state for that location. For example, in a 1-dimensional CA with two states and a neighborhood radius of two, a rule might contain the entry `11011` $\rightarrow$ `1`, which says that a location in state `0` surrounded by two `1`s to the left and the right will change to state `1`. 'Running' a CA involves taking an initial lattice and iterating for some number of time steps; the 'output' of the CA is the global state of the lattice when the system halts.

Despite their simplicity, CAs are remarkably interesting. For example, CAs can be Turing-equivalent [2], [1], [5]. Furthermore, CAs are good examples of complex systems, in the sense that they are composed of simple homogenous components (lattice locations) interacting locally (as determined by the extent of the rule neighborhood) to affect distributed information processing at the level of the entire system [3]. Consequently, studying the characteristics and behavior of CAs may provide insights into the nature of complex systems in general.

One problem that has received considerable attention is the "$\rho = \frac{1}{2}$" problem [4]. In this problem, the goal of a CA is to take an arbitrary initial lattice - an *initial conditon* (IC) - and, after some number of iterations, correctly classify the IC as having either more `1`s than `0`s or vice versa. In the former case, the CA should settle to a lattice of all `1`s, and in the latter the CA should settle to a lattice of all `0`s. $\rho_t$ is defined as the proportion of `1`s in the lattice at time $t$, so the problem can be stated as in (1), where $IC$ is an initial condition, and $n$ is the number of iterations.

$$\rho_n(IC) = \begin{cases} 0.0 & \text{if } \rho_0 < 0.5, \\ 1.0 & \text{if } \rho_0 > 0.5, \\ \text{undefined} & \text{otherwise.} \end{cases} \quad (1)$$

The "$\rho = \frac{1}{2}$" problem is interesting precisely because a global feature of the IC - i.e., $\rho$ - must be correctly categorized through only local information processing. Furthermore, the space of possible solutions is not small: For $k$ states and a neighborhood of radius $r$ there are $k^{k^{2r+1}}$ possible rules. So, for the parameters we consider in this paper ($k = 2$ and $r \in \{2, 3\}$), we are faced with the task of searching large sets of possible rules.

To cope with the size of the search space, we utilize a genetic algorithm (GA). In a GA, a relatively small number of possible solutions is randomly chosen from the set of all rules, and their performance is assessed against a random set of

ICs. A new set of rules is generated using mutation (i.e., rules are randomly modified) and crossover (i.e., parts from two rules are combined to form a new rule), where those rules that performed better on the ICs are more likely to be parents of the new rules. In this way, the GA randomly samples the space of possible solutions, and over multiple generations refines the search to look more closely at those regions that contain potential solutions.

In this paper we consider the capacity of a GA to locate solutions to the "$\rho = \frac{1}{2}$" problem, and also how successful those solutions are. In particular, we do the following:

1. In section 2.1 we compare the results of a GA search for rules with $r = 2$ and $r = 3$ on binary ($k = 2$) ICs of length 121. We characterize how the strategies of the best-performing rules change over the course of a GA run, and show that the $r = 3$ case outperforms the $r = 2$ case.

2. *Mutational robustness* occurs when phenotypic traits are invariant with respect to changes in the underlying genome [6]. In 2.2 we investigate whether successful rules exhibit mutational robustness by sampling additional rules in the neighborhood of successful rules. We find there is evidence in favor of robustness.

3. Finally, in 2.3 we attempt to improve on the results reported in 2.1 by dynamically adjusting the fitness function as a GA is executed so that it is gradually encouraged to search for solutions to harder ICs and discouraged from relying on previously found solutions to easier ICs. We show that this 'biased fitness' modification leads to better-preforming solutions, especially in the $r = 2$ case.

## 2. RESULTS

For the results reported here, the CA population is 100, selection is tournament with elitism, offspring CA populations are created with both mutation and crossover, the IC length is 121, the size of the IC test set is 100, and CAs are run for a maximum of 300 iterations per IC over 50 generations. See section 4 for more information regarding methods.

### 2.1 Comparing neighborhoods

Our first goal was to compare rules with different neighborhoods. In particular, there is a tradeoff between the size of the search space and the the amount of information available for purposes of setting a location state. Larger neighborhoods have rules that can 'see' more of the source IC, and hence make a more informed decision about how to set the output bit, but this comes at a cost of increasing the size of the search space by multiple orders of magnitude ($2^{32}$ vs. $2^{128}$).

Figures 1 and 2 show two typical CA runs. The first is an example of a rule that does not converge, while the second is a rule that successfully converges within 300 iterations.

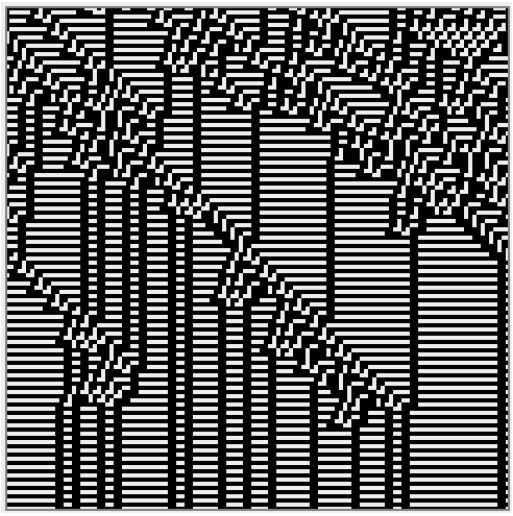

**Figure 1: Waterfall plot of a radius 2 rule that fails to converge within 300 iterations. The top row is the initial condition, and each successive row is the result of applying the rule to the row above it.**

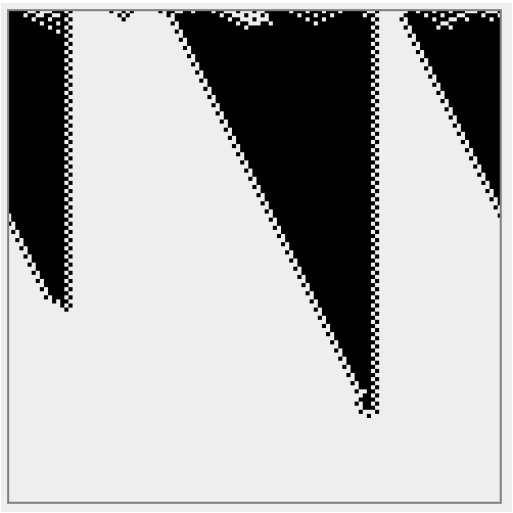

**Figure 2: Waterfall plot of a radius 2 rule that successfully classifies an initial condition.**

After running the GA for 50 runs, for both $r = 2$ and $r = 3$, we evaluated the elite rules from the final generation of every run against a test set of 10k ICs, and ranked them by their fitness score. Figure 3 plots the performance of the best performing $r = 2$ and $r = 3$ rules. As the figure shows, the best $r = 3$ outperforms the best $r = 2$ rule.

It is also interesting to compare the performance of the most fit $r = 2$ and $r = 3$ rules over multiple generations. Figures 4 and 5 show the best fitness for a sample run for the two neighborhoods as a function of generation. As can be seen in those figures, both trajectories exhibit 'epochs' as described by Mitchell [4]: There is an initial plateau interrupted by a

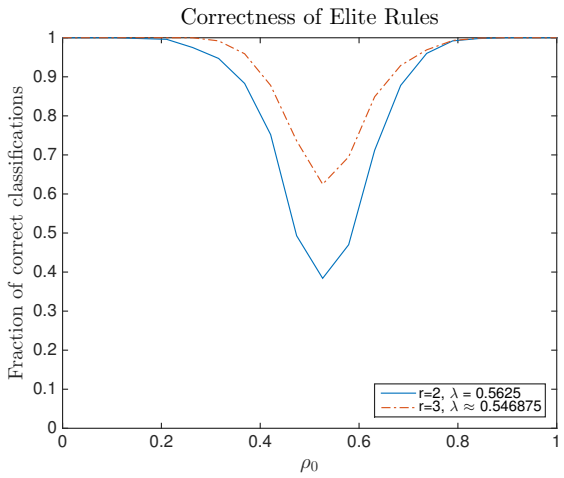


**Figure 3: The best performing elite rules from 50 GA runs each for $r = 2$ and $r = 3$. As expected, both rules perform well for extreme values of $\rho$, but performance drops as $\rho$ approaches the difficult cases around $\frac{1}{2}$. Furthermore, the $r = 3$ rule performed better than its $r = 2$ counterpart.**

jagged rise in best fitness, where these increases represent the discovery of new, dominant strategies.

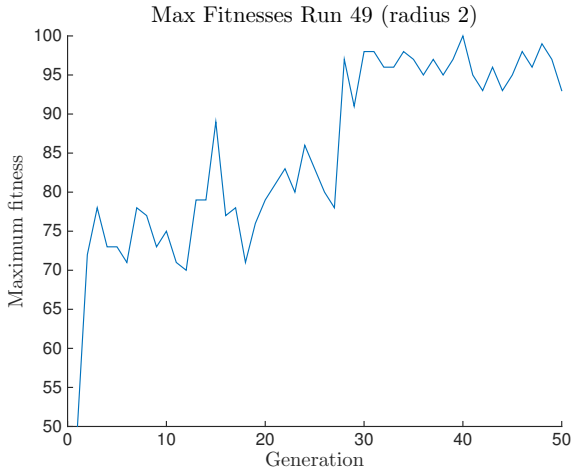


**Figure 4: Best fitness for an $r = 2$ run by generation. The maximum fitness trajectory exhibits epochs: the population rapidly leaves epoch 1, flattens out until about generation 30, and then climbs again to another plateau.**

Let $\lambda$ be the proportion of 1s in the binary representation of a rule [4]. Then the transitions just described are equivalent to shifting from rules with $\lambda \approx 0$ or $\lambda \approx 1$ to rules with $\lambda \approx \frac{1}{2}$. This transition is depicted in figures 7 and 8, for $r = 2$ and $r = 3$ runs, respectively.

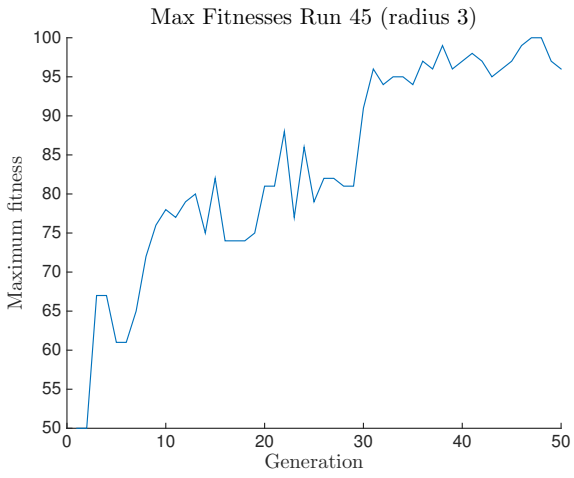


**Figure 5: Best fitness for an $r = 3$ run by generation. As in the case of $r = 2$ (fig. 4), the trajectory exhibits epochs; in this case, there appear to be three plateaus.**


**Figure 6: Average Shannon entropy of transients for 500 elite rules from the 50th generation of the $r = 3$ GA runs, as evaluated over a test set of 10k ICs. Elite rules do not favor short transients.**

Finally, note that each location in the lattice has a transient period in which its state may continue to flip before finally stabilizing (if at all; see figures 1 and 2). One way to quantify this period is to count the number of times a given location changes state over the 300 iterations of a rule, given an initial condition. In order to visualize the relation between the length of a transient period and the fitness of a rule, we recorded the average Shannon entropy of the transients for each elite rule on a set of 10k test ICs, and plotted this value against the fitness for that rule (figure 6).

## 2.2 Mutational robustness

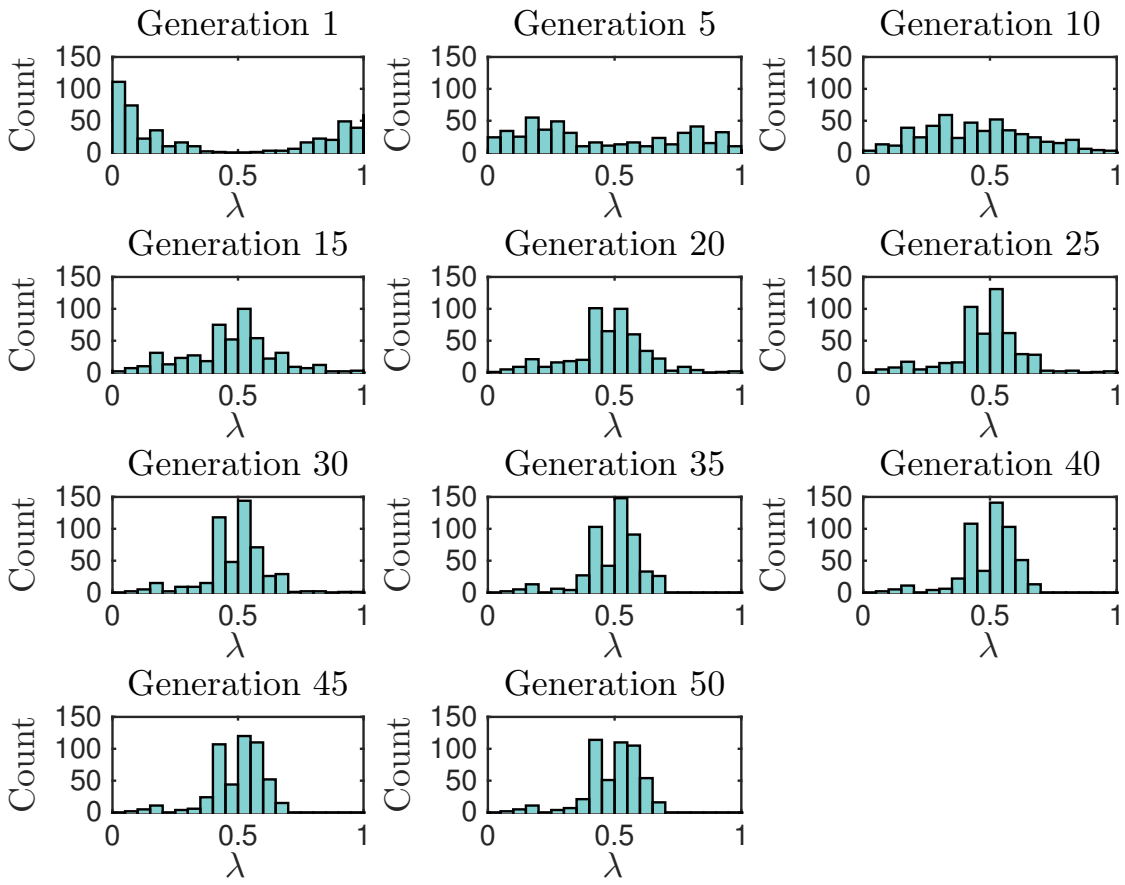


**Figure 7: Frequency of different elite rules (as represented by $\lambda$) by generation, for 50 $r = 2$ runs. Generations 1 and 2 are biased towards $\lambda \approx 1.0$ and $\lambda \approx 0.0$ indicating that successful rules converged on all 0s or all 1s, regardless of $\rho_0$. By generation 15 successful rules are starting to cluster near $\lambda = 0.5$, and we can also see the characteristic 'dual peaks' resulting from counting both rules that began life near $\lambda = 0.0$ and those that began near $\lambda = 1.0$.**

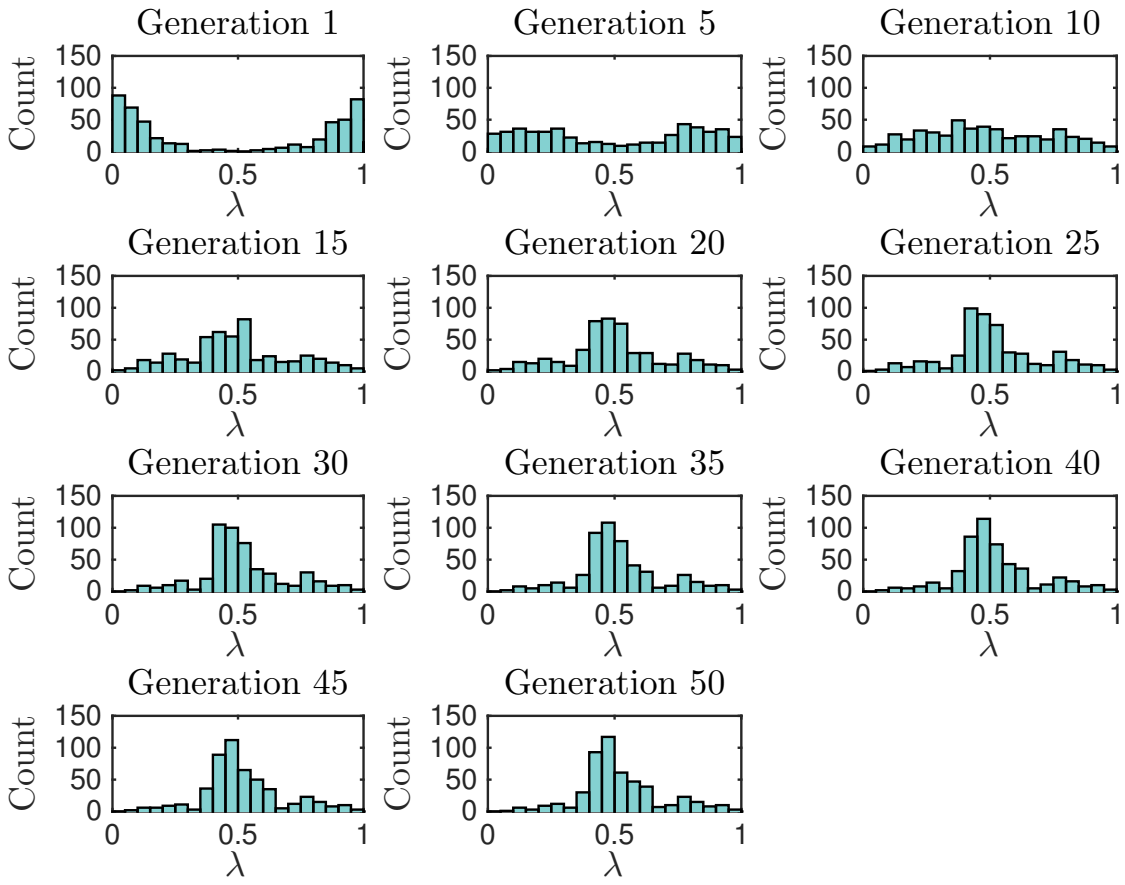


**Figure 8: Frequency of different elite rules (as represented by $\lambda$) by generation, for 50 $r = 3$ runs. Generations 1 and 2 are biased towards $\lambda \approx 1.0$ and $\lambda \approx 0.0$ indicating that successful rules converged on all 0s or all 1s, regardless of $\rho_0$. By generation 15 successful rules are starting to cluster near $\lambda = 0.5$, a trend that continues to the final generation.**

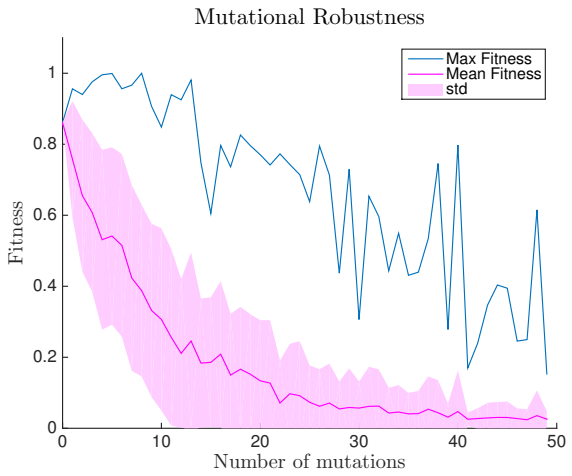


**Figure 9: Mutational robustness of the best performing $r = 3$ rule for 0 to 50 mutations. 100 unique variations of the original rule are generated with $m$ mutations, and each is evaluated over 10k randomly generated ICs. Mean fitness decreases rapidly, but high fitness rules are still present, as indicated by the max fitness values. Within a Hamming distance of $\approx 10$, maximum performance actually increases in comparison with the source rule.**

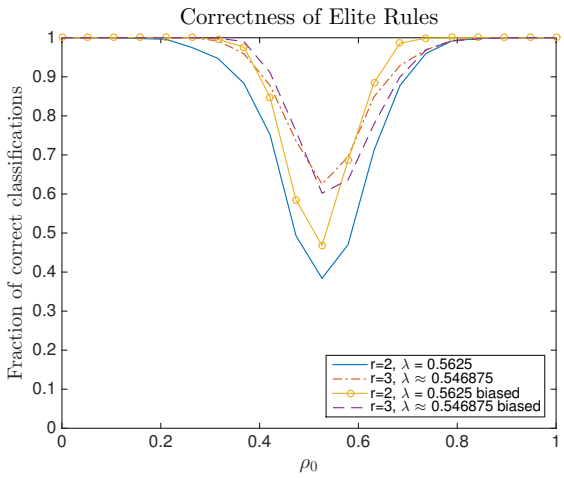


**Figure 10: The best performing elite rule from 50 runs of our 'biased fitness' GA, for both $r = 2$ and $r = 3$, plotted against the counterpart rules from figure 3.**

Mutational robustness occurs when changes in the genotype are phenotypically neutral: The genomic modification results in no significant change in the traits expressed, and hence no difference in fitness [6]. When a phenotype is robust in this way, it may be possible for the genome to move to a very different region of search space by accruing changes over multiple generations, increasing the breadth of the search without having to cross low-fitness boundaries.

If our CAs are mutationally robust, then the nearby neighbors of elite rules should themselves perform well. To assess whether this is the case, we tested the 500 elite $r = 3$ rules from generation 50 against 10k ICs, and selected the best performing rule. We then generated 50 sets of 100 mutated rules each from this source rule, with the first set containing mutants with a Hamming distance from the source of 1, the second a distance of 2, and so on. The fitness of each mutant was evaluated against 10k ICs; results are given in figure 9.

## 2.3 In search of a better CA: Biased fitness

As noted above, some of the earliest successful strategies to emerge are those that (i) go to all `1`s or all `0`s by default, yet (ii) change their behavior for values of $\rho$ at the opposite extreme. For example, a rule may for the most part map every neighborhood to `0`, except for neighborhoods with many `1`s, so an IC with a high enough value of $\rho$ will 'overwhelm' the default behavior of the rule. In such a rule, every IC with $\rho < 0.5$ is successfully classified, as are ICs with $\rho \approx 1.0$, but errors are made on those with $\rho \gtrapprox 0.5$.

Of course, the region around $\rho = 0.5$ is precisely the region we want to encourage the rules to deal with, yet under the fitness testing regime used for the experiments reported above, the 'easy' cases (where $\rho \approx 1.0$ or $\rho \approx 0.0$) appear in testing sets with the same probability as any other IC for the entire GA run. As a result, CAs are still being rewarded for dealing with these easy cases long after their utility is exhausted.

To address this issue, we modified the GA so that the fitness gain for successfully classifying ICs is a function of its distance from $\rho = 0.5$: The closer to 0.5, the more a correct classification is worth. Furthermore, this function is adjusted depending on the current generation, so that at generation 1 all ICs are worth the same amount of fitness regardless of their distance from $\rho = 0.5$, and the distance becomes increasingly important as the generations progress (see section 4).

We ran 50 runs of the 'biased fitness' GA, each for 50 generations, for both $r = 2$ and $r = 3$ cases. Best fitness by generation looks similar to that of the unbiased runs reported in section 2.1, so we do not give an example here. To find the best performing elite rule after the 50th generation, we tested each elite rule from each run on a test set of 10k ICs. Figure 10 shows the performance of the winning biased $r = 2$ and $r = 3$ rules when tested against 1000 ICs for each $0.0 \leq \rho \leq 1.0$ at 0.025 intervals; for purposes of comparison, the figure includes the plots from figure 3 as well.

# 3. DISCUSSION

There are a number of interesting observations regarding the results reported above; here we survey these conclusions.

## 3.1 On the difference between radius 2 and 3

The size of the search space grows with the radius of a rule ($2^{32}$ for $r = 2$, and $2^{128}$ for $r = 3$), suggesting that searches may be more difficult for CAs with larger neighborhoods. However, a larger radius also means that the CA has more information to work with in deciding which state to transition to, suggesting that finding a solution may be easier for

larger radii CAs despite the increased search size.

For both $r = 2$ and $r = 3$, we observe the pattern reported in [4]: Strategies begin simple and become more nuanced over time. We can see this progression in the series of histograms given in figures 7 and 8. In both cases, early populations (before generation 5) are dominated with rules that take an IC to all 0s or all 1s. This is a sort of 'gambling' strategy because it 'bets' that the correct answer is one way or the other. Interestingly, we also found rules that gambled in a different way: Oscillate $\rho$ back and forth from 1.0 to 0.0 in the hopes that, after 300 iterations, the final global state will be correct. Gambling is better than doing nothing, regardless of how you do it, and the GA was good at discovering this.

In both $r = 2$ and $r = 3$, $\lambda$ gradually moves towards 0.5: Rules retain the initial strategy of always converging one way or the other, but begin to include 'extreme' values of $\rho_0$ at the other end of the spectrum. However, we also see a slight difference in the histograms for $r = 2$ and $r = 3$. In the former, there is the characteristic 'dual peak' around $\lambda = 0.5$; this feature results from the fact that rules converge from both directions, i.e., from $\lambda \approx 0.0$ and $\lambda \approx 1.0$. But in the $r = 3$ case, we do not see this pattern; instead, we find a peak around 0.5 with troughs on either side. We are not certain what is responsible for this distribution, but the absence of dual peaks could be an artifact of the bin size for the histograms.

Addressing the issue posed at the beginning of this section, figure 3 shows that the best of the $r = 3$ rules performs better than its $r = 2$ counterpart. It thus appears that the larger search space of the former was not an impediment to finding a fit solution, and the greater amount of information available to the rule was a benefit.

Finally, our analysis of the relation between fitness and transients in elite rules is difficult to interpret. On the one hand, it seems that elite rules eschew short transients. On the other hand, the results also show that elite rules can perform terribly when tested against large sets of ICs. Qualitatively, when examining individual elite rules on an informal basis, we found that a surprising number were actually 'gamblers' of the sort described above. It may be that the low performers in the figure are present simply because the size of the testing set (100) is too small.

## 3.2 On mutational robustness

In section 2.2 we explored the fitness landscape surrounding the best-performing $r = 3$ rule discovered by our GA in an attempt to assess whether the rule exhibited mutational robustness, i.e., genomic modification without an impact to fitness. The fact that average fitness and its standard deviation decreases quickly (figure 9) shows that most mutations were deleterious to fitness. However, best-performing mutants actually did better for approximately a Hamming distance of 10 before starting to drop, and even then individuals occasionally performed well relative to the performance of the source rule.

This suggests that the landscape around the rule is 'rocky' in the sense that there are some paths that maintain (or even increase) a high fitness, while most do not. But the rule is not 'robustly robust' in the sense that any mutation is tolerated; there are some productive directions to go, but not every direction is productive.

## 3.3 On the quest for a better solution

ICs with extreme values of $\rho$ are easier to classify but interfere with the GA's ability to cope with harder cases at later generations. To address this, we implemented a 'biased fitness' version of the GA that (i) makes the fitness benefit of correctly classifying an IC a function of its distance from $\rho = 0.5$ (the further the distance, the less the fitness benefit), and (ii) has this bias increase as the generation increases (see section 4). In this way, the fitness function encourages the exploration of solutions to harder ICs as the GA proceeds.

Figure 10 shows that the performance of the best biased $r = 2$ rule outperforms its unbiased counterpart, suggesting that the dynamic fitness function succeeded in focusing the search on solutions to harder cases. The figure also shows that the best biased $r = 3$ rule appears to do about as well as its unbiased counterpart; however, these rules were selected using 10k random ICs, and the biased $r = 3$ rule outperformed the unbiased version on that test set.

# 4. METHODS

In this section we briefly describe the methods used to generate our results.

## 4.1 GA parameters

All runs of the GA used a population of 50 CAs, and the population is evolved for 50 generations. Initial rules were chosen from a uniform distribution of $\lambda \in [0.0, 1.0]$.

All CAs begin with an initial fitness of 0.0. At each generation, each CA was tested against a set of 100 ICs by running the CA on each IC for 300 iterations or until there was no change in the bit string over two iterations (indicating that the CA had settled into a steady state for that IC).

For the results reported in section 2.1, we used the fitness function given in equation 2, where $\Delta_{fitness}$ is the change in fitness for a CA, and $\rho_n$ is the iteration at which the CA stopped ($n \leq 300$). This fitness function is very simple: Given an IC, the fitness for a CA is increased by 1 if and only if it settles to all 1s when $\rho_0 > 0.5$ or settles to all 0s when $\rho_0 < 0.5$.

$$\Delta_{fitness} = \begin{cases} 1 & \text{if } \rho_0 < 0.5 \ \& \ \rho_n = 0.0, \\ 1 & \text{if } \rho_0 > 0.5 \ \& \ \rho_n = 1.0, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

With fitness function (2), the maximum fitness for a CA during its lifetime is 100, and the minimum is 0. Note that fitness scores are reset to 0 after each generation, even for the elite rules.

For the results reported in section 2.3, we adjusted the fitness function so that easy-to-classify cases (where $\rho_0 \approx 0.0$ or $\rho_0 \approx 1.0$) became worth less in comparison to ICs with

$\rho \approx 0.5$ as the generation increased (equation 3).

$$\Delta_{fitness} = \begin{cases} 1 + g\left(\frac{1}{|\rho - 0.5|50}\right) & \text{if } \rho_0 < 0.5 \ \& \ \rho_n = 0.0, \\ 1 + g\left(\frac{1}{|\rho - 0.5|50}\right) & \text{if } \rho_0 > 0.5 \ \& \ \rho_n = 1.0, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

In (3), $g$ is the current generation number, and 50 is a scaling term.

The GA uses tournament selection with elitism. After assessing the fitness of each CA, the top 10% are copied to the next generation, without any mutation or crossover. The remaining 90% are filled in by randomly selecting two pairs from the population, and the members of each pair compete for the opportunity to reproduce, the CA with the higher fitness wins. A single crossover point is chosen at random, and a new rule is constructed using one segment from each parent. Finally, 10% of the bits in the resulting rule are mutated (i.e., flipped).

## 4.2 Choosing ICs

Each generation was tested against a set of 100 ICs, and a new set was created for each generation. ICs were chosen from a uniform distribution $\rho_0 \in [1.0, 0.0]$ to make the inclusion of easier ICs equally likely as harder, $\rho_0 \approx 0.5$ ICs.

We considered changing this distribution for 2.3: Including the easier-to-classify ICs in the training set slows down the discovery of better solutions at later generations, so one way to correct for this is to make the inclusion of those ICs less likely as the generation increases. However, we instead opted for adjusting the fitness function as described above.

## 4.3 Implementation and Optimization

The GA was implemented in Java and JRuby with various optimizations. The individual computations of the CAs were identified as the dominant calculation performed by the simulation code. Optimizations were focused on those areas and in particular the following strategies were followed. Use of HashMaps for constant time lookups of rule behaviors to outcomes for the following bit string, use of just two byte arrays to represent the the previous state and the current state. A special java class was created to represent the the underlying rules behaviors so that HashMap usage would be an optimal match between reading sections of the current state of the CA and mapping to their outcome behavior in constant time.

All experiments were run on a late 2014 iMac, 4GHz, 4 core, Intel Core i7 with 32GB of RAM. The optimizations used have been benchmarked at a little over 350,000 CA iterations per second. This was combined with the Java Executor threading model, allowing an arbitrary number of threads to be launched and increasing the overall through-put on multicore systems. With this implementation we regularly achieved an actual rate of CA iterations of over 1,000,000 per second. A final optimization that greatly increased the effective rate of CAs per second was to stop processing a CA once it had arrived at a static state. This lead to effective processing rates of several million CA iterations per second and allowed for our code to be distributed across several individual computers to run multiple rounds of our GA code base (see nutella) if needed. However, the code ran quickly enough that a single 4 core system sufficed.

## 4.4 Data analysis methods

We made use of Matlab to analyze the data products of our code. Our automated programs were able to locate the best fitting rules that then allowed for direct analysis of the properties of these rules using Matlab scripts.

# 5. REFERENCES


[1] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays*. Academic Press, London, 1982.

[2] M. Cook. A Concrete View of Rule 110 Computation. *Electronic Proceedings in Theoretical Computer Science*, 1:31–55, June 2009. arXiv: 0906.3248.

[3] M. Mitchell. *Complexity: A Guided Tour*. OUP USA, May 2009.

[4] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Phys. D*, 75(1-3):361–391, Aug. 1994.

[5] P. Rendell. Turing universality of the game of life. In A. Adamatzky, editor, *Collision-Based Computing*, pages 513–539. Springer, London, 2001.

[6] A. Wagner. The role of robustness in phenotypic adaptation and innovation. *Proceedings of the Royal Society of London B: Biological Sciences*, 2012.