Pattern formation by evolution of cellular automata

Alex Garruss Casey Grun

May 8, 2014

1 Abstract

Biological systems are able to develop exquisitely complex patterns; the existance of very simple computer programs that can produce highly complex behavior suggest that the rules underlying these natural pattern formation processes may be very simple. In this work, we attempt to use an evolutionary algorithm to explore the space of initial conditions for an elementary cellular automaton (Rule 110), known to be Turing complete. We show that, despite the fact that the mapping between "genotypes" (initial conditions of a cellular automaton) and "phenotypes" (final states of the automaton) is complex, regions of local smoothness exist, and very simple patterns can be evolved by selection pressure. These results suggest that certain problems, despite having complicated genotype-to-phenotype mappings, may still be amenable to solutions by genetic algorithms if a fitness landscape can be formulated that is locally smooth. Finally, we discuss the applicability of these results to biological pattern-formation and general evolutionary processes.

2 Introduction

Elementary two-dimensional cellular automata are simple computational programs defined by a rule set and seeded with an initial condition state. The rule set determines the current value of a cell given a local neighborhood of previously computed cell states. The initial condition space can significantly impact the immediate computational context, drastically affect the overall state of the automata during computation, and ultimately determine the diversity of output states for a given fixed rule set and output definition. A traditional computer programming paradigm is frequently "top-down" in the sense rational paradigms and requirements of exactness for results (top-level outcomes) are directly achieved by design of lower-level procedures. In contrast, genetic programs as they exist through evolution tend to arise from the natural "bottom-up" paradigm, beginning with simple cellular interactions that form higher-order structures and patterns. Cellular automata provide a computational paradigm to explore simple rules and resulting complexity. Cellular automata states are abstract and allow granularity corresponding to any part of an object or process. Rule 110 was shown [1] to be Turing complete by showing equivalence of the automata to cyclic tags, previously proven to be complete. This feature of the rule set implies a capability to harbor a vast diversity of output configurations, and potentially capable of harboring structures or patterns within the computed landscape of the running automata.

Emerging studies over the past several decades have looked to various cellular automata to describe patterns in biology. In 1999 the theoretical biologist Andreas Deutsch proposed a migration automata to model cellular movement and morphogenetic pattern formation. [2] After building and refining a model (although not an elementary CA as proposed in the project), he describes the process of inferring simulated outcomes from observing rule definitions themselves. For our proposed work, we expect to learn information about selected automata rule sets or initial conditions after computational evolution. Deutsch proposes further morphogenetic description possibilities via simple automata to investigate other areas of

pattern formation such as body formation. We agree with his proposition, noting the sequential, hierarchical, and spatially-local pattern formation of the fly embryo. [3] Furthermore, Deutsch suggests extending the model to link proper pattern formation to subsequent cell decisions, such as differentiation. This suggestion could be interpreted in our proposed framework to shift rule sets upon a resulting pattern of interest during computation.

Other groups (reviewed by Ermentrout et al. [4]) have also approached pattern formation, along with areas in neurobiology and population dynamics. Our approach differs dramatically in most cases since most of the previous groups rationally target mathematical models for a task at hand, whereas we propose to evolve input conditions to an elementary cellular automaton in an non-rational, unsupervised way.

Our project is to use an evolutionary algorithm to explore the process of computational pattern formation using cellular automata. We take inspiration from Stephen Wolframs program to exhaustively explore the behavior of each possible rule set for an elementary cellular automaton, [5] and would like to perform a similar such exploration of the space of initial conditions for such an automaton. Rather than exhaustively enumerate and then examine all possible initial conditions, we will take a restricted, evolutionary approach. Specifically, we will attempt to answer the question: can we use evolutionary algorithms to generate initial conditions ("genotypes") for a cellular automaton that can produce arbitrary output patterns ("phenotypes")? Our starting point is Rule 110.

There are several important challenges in addressing this question:

Formulation of the evolutionary process — It is not immediately obvious how to map cellular automata on to a genotype space which can be evolved; similarly, how should one perform recombination on the features of the genotype space? We explore several models (see Methods below) for the genotype and fitness evaluation of our cellular automata.

Discontinuity of fitness landscape — A fundamental premise of evolutionary search algorithms is that there exists some fitness process that can be expressed as a function of the genotype of the objects under selection, and that this function is (mostly) smooth. That is, small changes in the genotype should not (usually) produce large discontinuous jumps. However, in our case, the function mapping genotype (initial conditions) to fitness is in principle not smooth—since Rule 110 can simulate a Turing machine, the function mapping initial conditions to output conditions (and in turn to the fitness) is undecidable (if the lifetime of the automaton is not fixed). Therefore, one motivation for this project is a search for a locally smooth region of this highly discontinuous fitness landscape—are there initial conditions such that small perturbations produce little variation in the output?

Universal computation $\neq$ arbitrary patterns — It is not obvious to us what class of patterns can be formed by an automaton running Rule 110, given various initial conditions, or what modifications to the automaton may be necessary and sufficient to allow formation of arbitrary patterns. It does not seem likely that, just because Rule 110 is Turing universal, it can also produce arbitrary output patterns. It does seem possible that arbitrary pattern generation could be possible with, for instance, a fixed basis function transformation on the output vector. We have attempted to explore and understand the space of patterns that can be formed by these types of automata.

In this paper, we develop a simulator for Rule 110 cellular automata and a general evolutionary algorithm solver, and we use this software to explore the evolution of Rule 110 inputs for pattern formation. Additionally, we use the data sets from our pattern-formation experiments to explore the local smoothness of the fitness landscape for these various evolutionary processes. We find that it is possible (though difficult) to evolve initial conditions that perform very rudimentary pattern formation using relatively short automata run times. We show that for these types of processes, some local smoothness

of the fitness landscape can be observed. We discuss the applicability of these findings to biological pattern formation and to evolutionary algorithms and processes more generally.

3 Methodology

3.1 Cellular Automata

Although cellular automata have been implemented before in packages such as Mathematica, we chose to implement a basic two-dimensional cellular automata in C++ from the ground up. Our choice to build our own version was based on considerations for performance and ease of parameterization. We wanted to avoid the overhead of interfacing our genetic algorithm with another stand-alone automata that may introduce delays in frequent instantiation. In our production system used for simulation, a genome size of 100 bits, using a population or pool size of 100 individuals, executing the cellular automata for 100 steps, and while operating under our full genetic algorithm for 100 generations, completed processing in around one minute. This relatively quick turnaround, however, increases to five minutes with a genome size of 1000 bits running for 100 generations, and to just under an hour for a 1000 bit genome running for 1000 generations. In our running automata, update decisions for a current generation are computed in simple way of explicitly determining the previous time point neighborhood.

For each $i \in I$, where I is the genome size:

```
// previous 111 110 101 100 011 010 001 000
// new state 0 1 1 0 1 1 1 0

if(
  (w[i-1]==0 && w[i]==0 && w[i+1]==1)  ||
  (w[i-1]==0 && w[i]==1 && w[i+1]==0)  ||
  (w[i-1]==0 && w[i]==1 && w[i+1]==1)  ||
  (w[i-1]==1 && w[i]==0 && w[i+1]==1)  ||
  (w[i-1]==1 && w[i]==1 && w[i+1]==0))

{  x[i] = 1; } else { x[i] = 0; }
```

...where x is the current vector and w is the previous time step vector. We primarily used text based output of our genotypes and final automata state (phenotypes) to rapidly prototype and debug our system. Evaluation of phenotypes was simplified in our system since the current state of the automata is already in memory and need not be retrieved from a stand-alone package. Performance gains are likely possible with optimizations to our core update algorithm, since it explicitly queries the previous time step neighborhood for each new cell, leading to redundant calls for roughly 2/3 of the previous time step vector. Utilization of efficient data structures such as quad trees might be an additional way to speed up the automata engine portion of our system.

3.2 Evolutionary Algorithm

We implemented an general evolutionary algorithm in C++ to select a pool of n genomes according to fitness. At each evolutionary iteration, the algorithm would: mutate all genomes, kill off (assign fitness of $-\infty$) any genomes older than a certain age, determine the "phenotype" of each surviving genome, and evaluate the fitness of each phenotype. Then it would sort the genomes in order of decreasing fitness; the m "fittest" genotypes would remain in the pool unchanged, while the bottom $n-m$ genotypes would be replaced by genotypes generated by recombining two random genotypes from the surviving pool. This process is summarized in Alg. 3.2

In our representation, the genomes were each bit vectors of length ℓ. Mutation occurs choosing $\phi * \ell$ bit(s) in a genome uniformly randomly (that is, some fraction ϕ of each genome will be subject to mutaton), then independently setting each of those bits to a random value, 0 or 1. Recombination occurs by choosing choosing some uniformly random crossover point $\ell' \in \{1 \ldots \ell\}$, then concatenating the first ℓ' bits from one parent with the last $\ell - \ell'$ bits from the other parent.

The fitness of each genome is evaluated by running a cellular automaton for a fixed number of time steps, with the genome sequence (the "genotype") as the initial state. The final state of the automaton is the "phenotype." The phenotype is compared by Hamming distance with some target bit vector (also of length ℓ); the fitness is given as the negative Hamming distance between the phenotype and the target

```
1: procedure GeneticAlgorithm(pool size n, generation size m, generations s)
2:     G ← ⟨n random genotypes g_1 ... g_n⟩
3:     F ← ⟨f_1 ... f_n⟩
                          ▷ Initialize gene pool, fitnesses
4:     repeat
5:         for all i ∈ {1 ... n} do
6:             Mutate genone g_i
7:             if g_i older than threshold then
8:                 f_i ← −∞
9:             else
10:                φ_i ← phenotype of g_i
11:                f_i ← fitness of φ_i
12:            end if
13:        end for
14:        Sort gene pool G by fitnesses f_1 ... f_n
15:        for all i ∈ {m ... n} do
16:            Choose two random genotypes
17:            j, k ∈ {1 ... m}
18:            g_i ← Recombine g_j, g_k
19:        end for
20:    until Stopping condition is met
21: end procedure
```

bit vector (such that greater fitnesses are preferable).

3.3 Fitness Landscape Investigation

We investigated how changes to a genotype correlated with changes to the phenotype and fitness of that genome. Consider a gene pool $G = \{g_1, g_2 \ldots\}$, where g_i are bit vectors. Each genotype g_i has some phenotype ϕ_i, which is also a bit vector. To determine the smoothness of the phenotype landscape, we compute a vector of genome distances $\hat{g}_i = \langle \hat{g}_{i,1}, \hat{g}_{i,2} \ldots \rangle$ such that $\hat{g}_{i,j} = \textsc{Hamming}(g_i, g_j)$ (where $\textsc{Hamming}(\cdot, \cdot)$ represents the Hamming distance), and similarly a vector of phenotype distances $\hat{\phi}_i = \langle \hat{\phi}_{i,1}, \hat{\phi}_{i,2} \ldots \rangle$ such that $\hat{\phi}_{i,j} = \textsc{Hamming}(\phi_i, \phi_j)$. We then computed the Pearson correlation $\rho_i^{\phi} = \text{Corr}\left[\hat{g}_i, \hat{\phi}_i\right]$, for each $g_i \in G$, for each generation G.

We performed a similar analysis to determine the smoothness of the fitness landscape; we computed a vector $\hat{f}_i = \langle \hat{f}_{i,1}, \hat{f}_{i,2} \ldots \rangle$, where $\hat{f}_{i,j} = |f_i - f_j|$. We then computed the correlation $\rho_i^f = \text{Corr}\left[\hat{g}_i, \hat{f}_i\right]$, for each $g_i \in G$, for each generation G.

The correlations ρ serve as measures of smoothness, because in a smooth landscape, small changes to the genotype would produce proportionally small changes in the fitness (and similarly for the phenotype). In principle, a smooth landscape should produce values of ρ_i^f (and ρ_i^{ϕ}) close to one—for instance, if the genotype-to-phenotype function were simply the identity, then $\rho_i^f = 1$ for all g_i. On the other hand, a very rough landscape would have values of ρ_i^f close to zero (such that small changes and large changes to the genotype are equally likely to produce changes of varying magnitudes).

4 Results

4.1 Evolutionary Pattern Formation

Our initial simulations used a target of all 0's to get a feel for the evolvability of input conditions for our cellular automata. By running our automata for only one step, we tuned the performance of the genetic algorithm to find phenotypes that were very close to

our blank target. We introduced several improvements our system performance toward reaching an all blank input condition that results in a blank automata. In condition to the features we continually improved in the genetic algorithm, such as random cross over position, we also added a way to protect a small number of the top performing genotypes from mutation. We found that when mutation affects all genotypes, arriving at a initial condition of all zeroes is nearly impossible. Protecting several genomes from mutation helped our system arrive at blank input states. Even when running for 100 steps, nearly all sampled simulations were able to find the all blank initial condition for the blank target.

We next turned our attention to varying target patterns. Three new patterns were chosen, which we call: alternative, half, and wave. The alternating pattern presents a series of one set bit (a 1) followed by one 0 bit, such as 1010101010. The half pattern presents half of the phenotype as 1's and the other half 0's. And the wave pattern presents a series of ten 1 bits followed by ten 0 bits for the length of the desired phenotype. Using the half pattern and a relatively short cellular automata runtime of 10 steps, we investigated the effect of generation time, or rounds of evolution, on the maximal fitness of evolved input states (Fig. 1). As expected, longer generation times had the increased probability to find better performing genotypes.

Using long-running simulations of 1000 generations, we looked for fitness differences in the total number of columns of our cellular automata, or variable genome sizes (since the genotype and phenotype are the same size in our system). Interestingly, larger genome sizes performed worse at evolving our new targets than the genome sizes for 10 and 50 (Fig. 2). Furthermore, we asked how increasing the pool size, or number of individual genomes in the population, affected performance. Larger pool sizes has increased probability of having highly fit genotypes out of the gate, and generally had steeper slopes of fitness gains per generation (Fig. 3). By the last generation, larger pool sizes maintained an advantage of the smaller pools.

4.2 Fitness Landscape Smoothness

We observed that, in general, the fitness landscape is not smooth. This was as expected—small changes in the input to the cellular automaton may produce chaotic changes in the output of the automaton. This can be observed by considering two simple example genotypes: one genome containing all 0 bits and one genome containing a single 1 bit. The former will produce a phenotype containing only zeros, while the latter will produce a complex phenotype.

We did observe two interesting phenomena, however: first, there are pockets that are "smoother" than others. Fig. 4a–b demonstrate that smoothness (as measured by ρ_i^f) is highly non-uniform across different genotypes and different generations. Fig. 4a shows the distribution of smoothness values across all genomes considered in the evolutionary process. In particular, note that the mean smoothness is not 0, but $\approx$0.2, and that the tails of the distribution extend to $\pm$0.4. This skewness towards positive values for ρ_i^f provides initial evidence that there are locally-smooth regions of the fitness landscape. Fig. 4b shows that smoothness values are not distributed uniformly-randomly—there are structural patterns both at the level of individual generations, as well as across generations. Within generations, the fittest genotypes (furthest-left) tend to be more smooth; this makes some sense, since it suggests that the fittest genotypes may have a certain robustness to small mutations. Across generations, it is possible to observe episodic patterns (black brackets) in the smoothness of the resulting landscape—in the bracketed regions, correlations tend to be polarized towards values of $\pm$0.4, whereas in un-bracketed regions, the correlations are closer to zero. These phases may correspond to periods of broad vs. deep exploration—in locally smooth regions, most mutations produce small changes in fitness, whereas in locally rough regions, small mutations can produce large swings in fitness.

Second, when examining plots of genome distances $\{\hat{g}_{i,j}\}$ vs. fitness differences $\left\{\hat{f}_{i,j}\right\}$ for various genomes (Fig. 4c), we observe direct evidence of locally-smooth regions. Specifically, the appearance of points where small changes in the genome yield

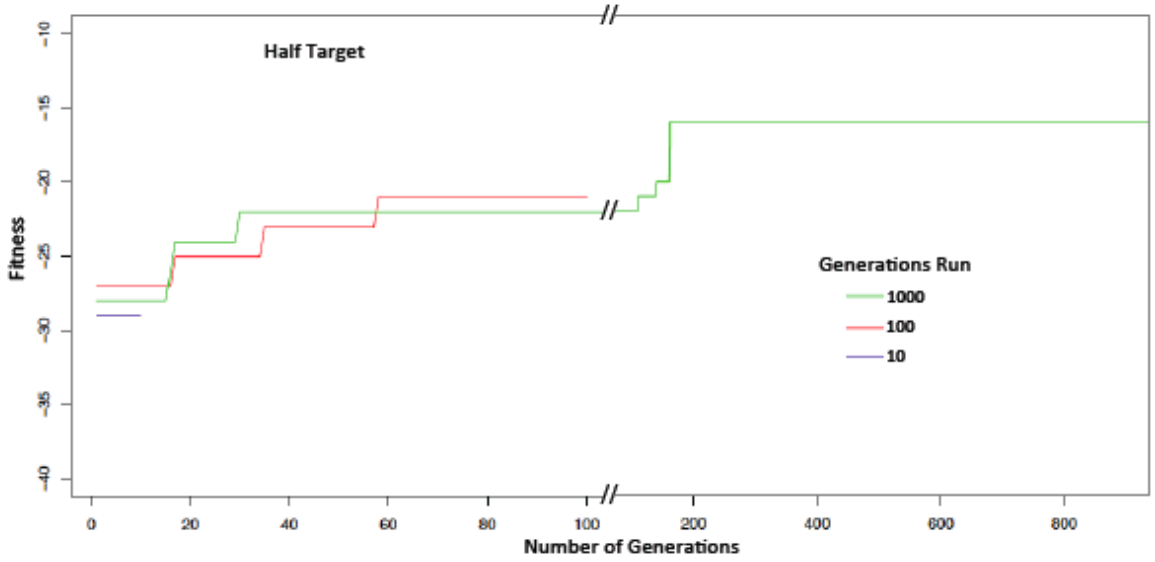

Figure 1: Affect of generation length for three attempts to find the "half" target. Identically parameterized runs differing by the number of rounds of evolution (generations). The genome size is 100 bits, pool size of 100 individuals, run for 10 cellular automata steps. The "half" target consists of half 1's and half 0's. The y-axis shows the number of bits in the final condition that differ from the target. Longer generation times are more likely to have bursts of improvement, such as between steps 100 and 200 for the longer running population.

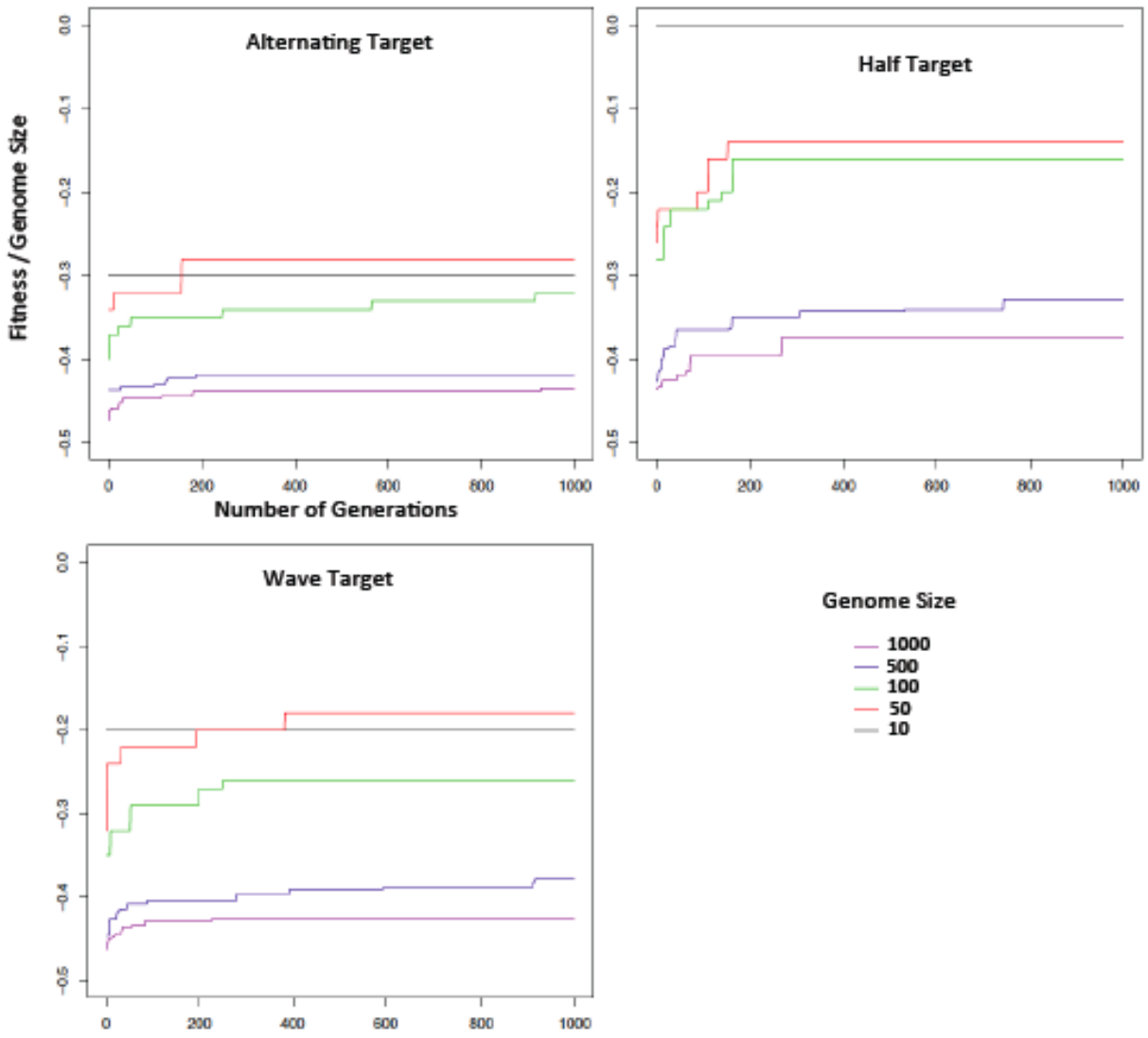

Figure 2: Affect of genome size on performance of different targets. Five different genome size lengths were compared to three targets: alternating, half, and wave. Larger genome sizes generally have more difficulty resolving the target after 10 steps of automata. All simulations were run for 1000 generations. The y-axis reflects the corresponding percentage of difference between the best performing phenotype to target for each generation.

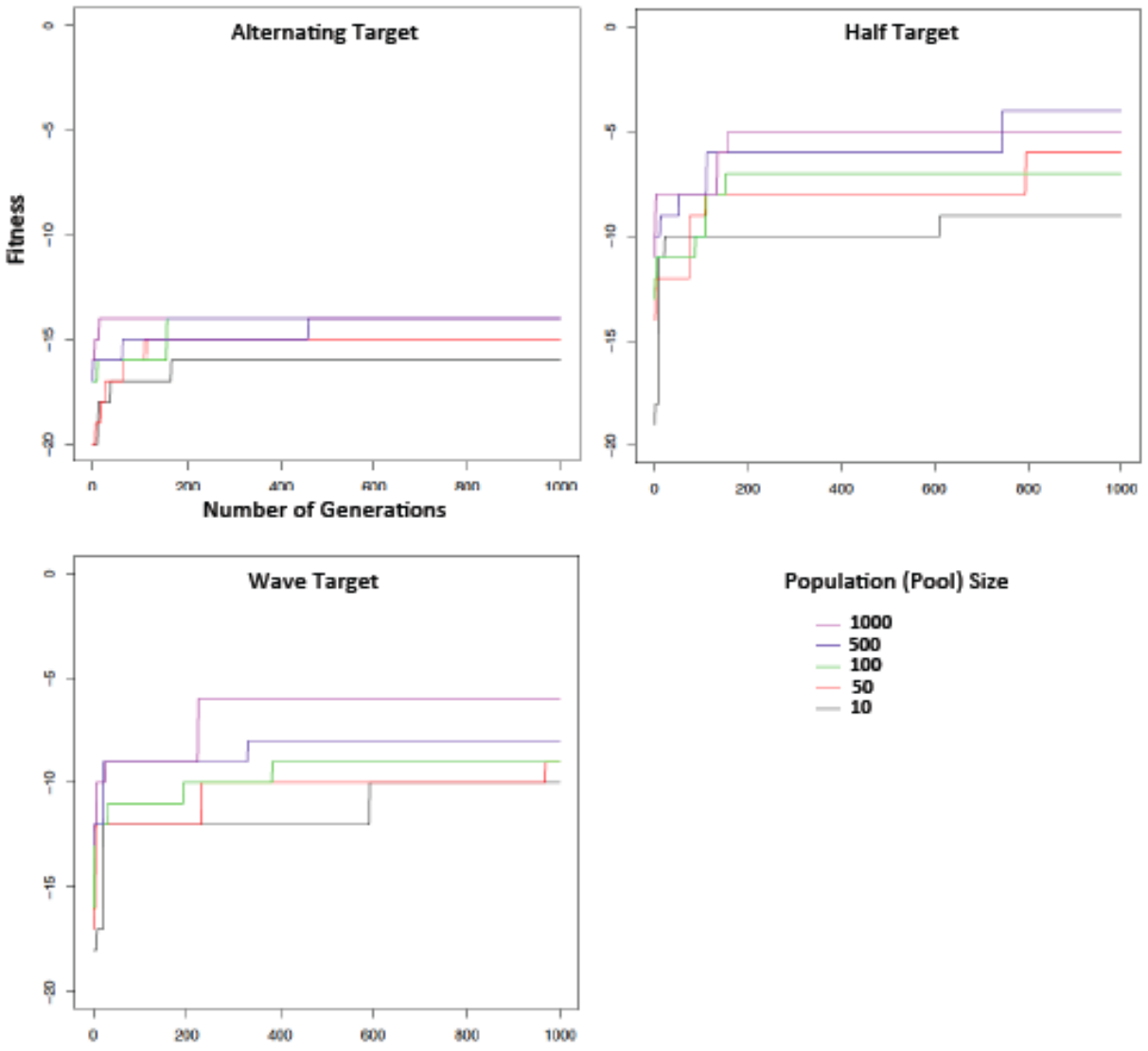

Figure 3: Affect of pool size on performance of different targets. Varying population or pool sizes were compared to the three targets. Larger pool sizes generally have more success resolving the target after 10 steps of automata. All simulations were run for 1000 generations. The genome size used was 50 bits.

small differences in fitness suggests that the fitness landscape is in fact smooth at those points, in certain directions.

5 Discussion

All simulations were capable of finding a highly fit genotype to match a blank target, no matter how many steps of the automata were run. Performance for the other targets, however, was more difficult for the system to evolve than expected. For the alternating pattern, larger pool sizes generally and longer generation times had little effect on performance, indicating it just be a really hard target. For the half pattern, most simulations continued to make progress over very long runs but benefited from a smaller genotype size. The finding that a smaller automata space improves performance may be an underlying feature of the rule set. As population sizes increase, so does the probability of finding a more fit genotype; however, longer rounds of evolution can help make up this advantage, as shown in the half pattern case where even a pool size of 50 approaches the max fitness of 1000 after hundreds of rounds. A thorough investigation of the trade-offs between large populations and long generation lengths under limited resources might be highly beneficial for future studies.

The question of fitness landscape smoothness is central to the question of whether a particular search problem is amenable to solution by an evolutionary algorithm at all. We observe that the fitness landscape defined by our evolutionary processes, while not generally smooth, has regions of local smoothness. On the one hand, this is surprising, given the chaotic nature of Rule 110 and the undecidable mapping between inputs and outputs for generalized 110 processes (e.g. cellular automata running Rule 110, whose stopping conditions are determined by automaton output). On the other hand, biological systems are constantly solving an evolutionary problem with a highly non-smooth fitness landscape—many random mutations are lethal or result in drastic changes in fitness. Epistasis (multiple mutations having non-additive effects) and pleiotropy (individual genes influencing multiple, unrelated phenotypic traits) create the potential for significant jumps in fitness with small mutations to a genome. [6] However, certain properties of the genotype-to-phenotype mapping (for instance, degeneracy in the genetic code, allowing for mutations that produce identical or similar peptide sequences) allow for a high degree of local smoothness. Further, features of the selection process allow populations of various sizes to cross fitness valleys, even in the absence of sexual reproduction. [7] Our observation suggests that certain search problems, despite having a complex mapping from genotype-to-phenotype, may have simple formulations that allow for "smooth enough" fitness landscapes for evolutionary search.

6 Conclusion

There is room for much further work in the exploration of the shape and smoothness of the fitness landscape. A more systematic investigation of the genotypes with locally smooth neighbors could yield interesting insights into the space of possible Rule 110 inputs. We have made qualitative observations about the patterns of smoothness within and across generations (e.g. that high-fitness genomes exist in smoother regions, and that episodic behavior occurs in the smoothness of generations); it would be interesting and fruitful to test these hypotheses by bringing statistical tests to bear on larger experiments than were attempted here.

7 Contributions

C. G. wrote the code for the evolutionary algorithm, and performed the analysis of the fitness landscape smoothness. A. G. wrote the code for the cellular automaton and performed simulations to test evolutionary pattern formation.

References

[1] M. Cook, "Universality in elementary cellular automata," *Complex Systems*, vol. 15, no. 1, pp. 1–40, 2004.

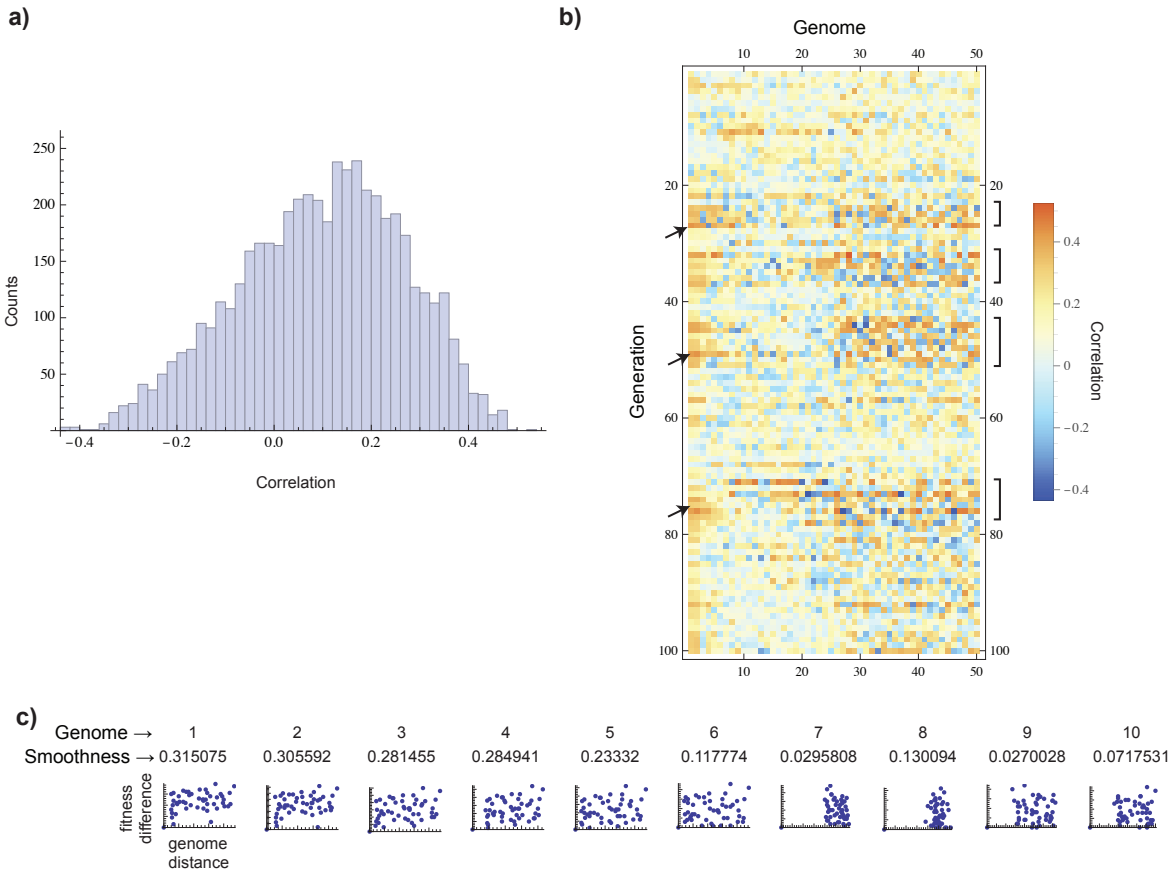

Figure 4: Smoothness of Fitness Landscape. **(a)** Histogram showing the distribution of values for ρ^f over all genomes in all generations of an evolutionary process. **(b)** Plot showing ρ_i^f for each genome in the first 100 generations of an evolutionary process; each row is a different generation, and cells within a row are distinct genomes, ordered left-to-right by decreasing fitness. Colors show the values of ρ_i^f for each cell. **(c)** Series of plots comparing genome distances $\hat{g}_{i,j}$ to fitness distances $\hat{f}_{i,j}$, for $i = 1, 2, \ldots 10$ (the 10 genomes with the highest fitnesses) in generation #77.

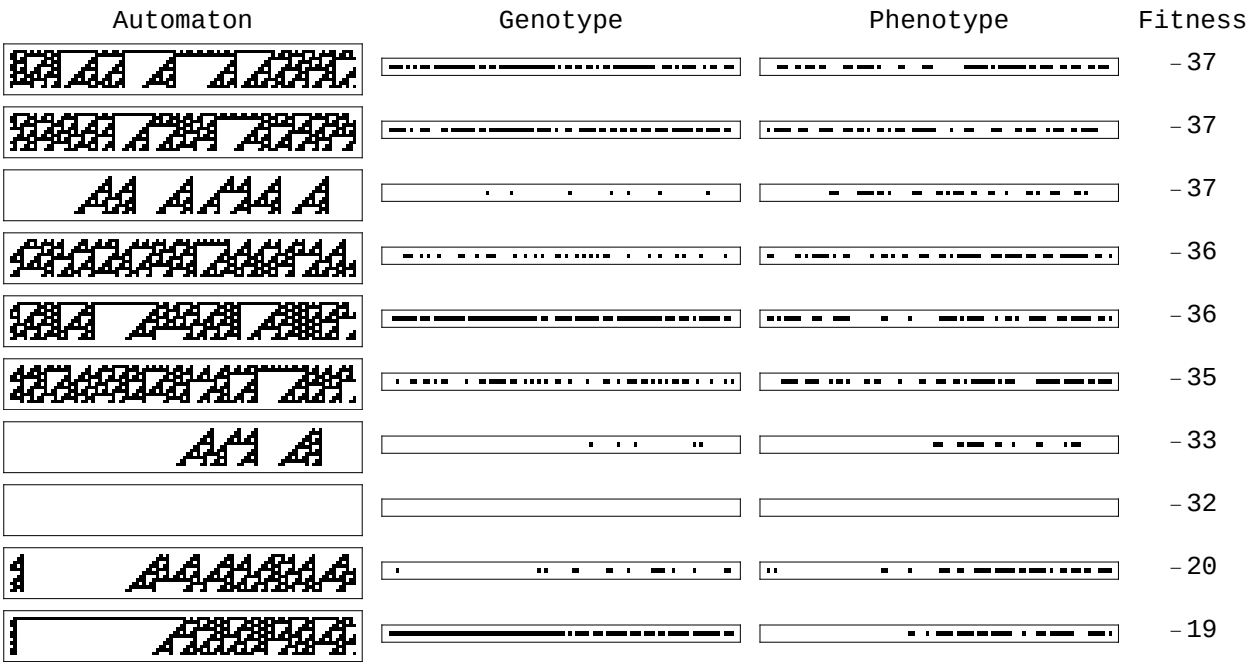

Figure 5: Different genotypes with similar performances. The 10 fittest genotypes in an evolutionary process whose target structure was half 0's, half 1's. In this process, the genotype-phenotype mapping was achieved by running the automaton for 10 steps. These genotypes are selected from a pool of 100 genotypes, evolved for 1000 selection cycles.

[2] A. Deutsch, "Principles of biological pattern formation: swarming and aggregation viewed as self-organization phenomena," *Journal of biosciences*, 1999.

[3] D. E. Clyde, M. S. G. Corado, X. Wu, A. Paré, D. Papatsenko, and S. Small, "A self-organizing system of repressor gradients establishes segmental complexity in Drosophila.," *Nature*, vol. 426, pp. 849–853, Dec. 2003.

[4] G. B. Ermentrout and L. Edelstein-Keshet, "Cellular automata approaches to biological modeling.," *J. Theor. Biol.*, vol. 160, pp. 97–133, Jan. 1993.

[5] S. Wolfram, *A New Kind of Science*. Wolfram Media Inc, Jan. 2002.

[6] B. Ostman, A. Hintze, and C. Adami, "Impact of epistasis and pleiotropy on evolutionary adaptation.," *Proc. Biol. Sci.*, vol. 279, pp. 247–256, Jan. 2012.

[7] D. B. Weissman, M. M. Desai, D. S. Fisher, and M. W. Feldman, "The rate at which asexual populations cross fitness valleys.," *Theor Popul Biol*, vol. 75, pp. 286–300, June 2009.