EVOLUTION OF COMPUTATION

JAYANTH GARLAPATI, SIMON DEDEO, JOHN MILLER

1. Introduction

Computer science is the study of discrete information processing and computation. These ideas are usually employed in the context of silicon machines, but can we think of natural systems as computing and understand their functioning in terms of their computational structure? With the abundance of experimental data on biological systems, it has become possible to probe these questions more concretely [1]. To understand biological systems in terms of computation, first we must describe them in the formalism of mathematics and computer science [2]. Within this formalism, we must define what we mean by computational structure. Finally, we can relate the function of the system to its computational description. One approach is to use the formalism of finite state automata to describe biological systems [3]. Finite state machines are simple machines that have states and take inputs that dictate transitions between states. They are very general and can be used to describe a variety of systems and agents.

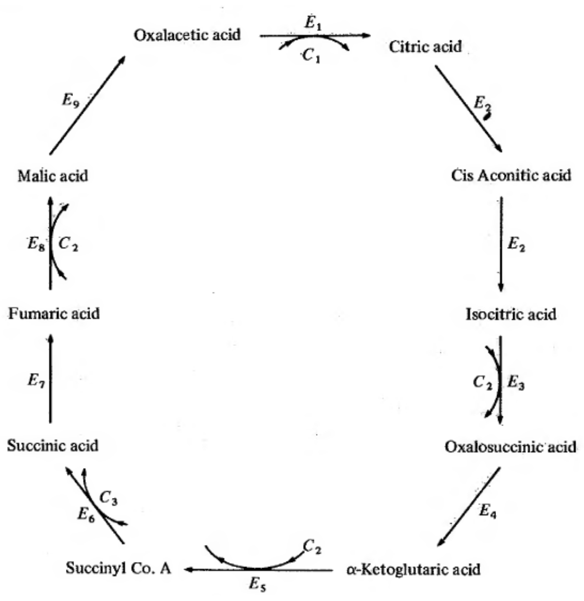

Figure 1. Finite State Automaton modeling the Krebs Cycle

Finite State Automata give an abstract, mechanistic description of a system, but it is not necessarily clear what computation is in such an operational diagram. Understanding computation in an automaton is further complicated in large systems for which automata descriptions can become rather complex. We follow [4] and [5] and represent finite state machines as semigroups so we may employ algebraic structure results such as the Krohn-Rhodes Decomposition theorem to identify and study underlying symmetries and computational structure in systems. Furthermore, we seek to understand how such computational structure arises under the pressures of evolution. To explore these questions, we consider work done in evolutionary game theory [6], where automata learn to play a game under the pressures of evolution, from this algebraic perspective to understand the computational structure of the machines playing games and how such structure arises through evolution.

2. Finite State Automata and Semigroups

Definition 1. *A Deterministic Finite State Automaton (DFA) A is given by a tuple $A = (Q, \Sigma, \delta)$ where* [7]

- *Q is a set of states*
- *Σ is a set of input letters, the alphabet. We denote the words formed by the alphabet with Σ^+*
- *$\delta : Q \times \Sigma \to Q$ is a transition function, which determines what state the machine transitions given an input and the current state*

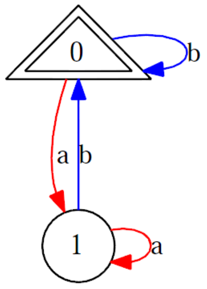

Figure 2. Two state DFA defined on two letters

DFA are often represented as graphs as in figure 2, but as their structure gets large, it can be difficult to understand their functional structure. Additionally, graphical analysis or inspection can be insufficient to understand the underlying structure of the machine. We instead represent DFA as semigroups to better understand their transition structure.

Definition 2. *A Semigroup $(S, \cdot)$ is a set S with an associative binary operation $\cdot$*

Example 1. $(\Sigma^+, \cdot)$ *is the semigroup of words generated by the letters in Σ with concatenation*

To represent a DFA as a semigroup, first we extend the domain of the transition function δ to take words from Σ^+ as inputs inductively in the following manner $\delta(q, ab) = \delta(\delta(q, b), a)$. Each $s \in \Sigma^+$ induces a mapping $\delta_s : Q \to Q$ where $\delta_s(q) = \delta(q, s)$ [8]. Notationally, we refer to the mapping δ_s by the word s and we say s acts on q as follows $s \cdot q = \delta_s(q)$. The Transformation semigroup (TS) of a DFA $A = (Q, \Sigma, \delta)$ is $(S, Q) = \langle \delta_s | s \in \Sigma^+ \rangle$ i.e. all the unique transformations of the state set Q induced by words in Σ^+. The transformation semigroup of a finite state automaton entirely describes the transition structure of the machine.

Definition 3. *A word $s \in \Sigma^+$ is a permutation if $s \cdot Q = Q$ and s is a reset if $s \cdot Q = q$ for some $q \in Q$. A DFA A is a permutation automaton if every element of its transformation semigroup is a permutation and a reset automaton if every element of its transformation semigroup is a reset.*

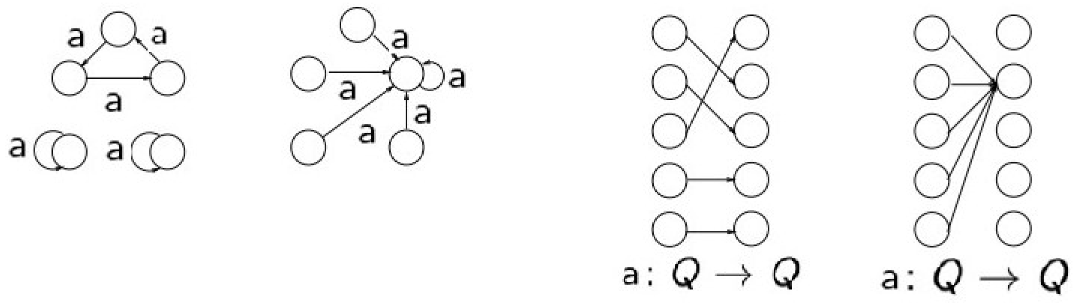

Figure 3. Example of Permutation and Reset Automaton

A permutation is invertible so we say that it is a reversible computation; whereas when a reset is applied, all state information is lost, so we say that it is an irreversible computation [9]. Since a permutation has an inverse and is a finite mapping, it generates a group in the transformation semigroup of an automaton. We can think of a group in the TS of a DFA as a type of computation or counter. For example in figure 4, the input letter C in the machine is a permutation and generates the group $\mathbb{Z}_3$ in the TS of the machine and is a means for the machine to count to three.

In practice, the counting and computation implied by a group can be much more complicated. For instance, cyclic groups of composite order can be thought to count by multiple increments. Non-abelian groups can distinguish between the sequence of inputs in addition to counting. It is still unclear what the role of Simple Non-Abelian Groups (SNAGs) is computationally, but some believe that such groups have interesting and important implications for the computation of a DFA [5].

3. Krohn-Rhodes Theorem

We can think of the evolution of computation in terms of the changes of group structure in the TS of a DFA. However, automata can have a very large semigroups. For instance,

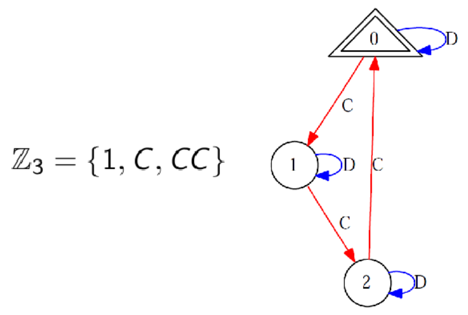

FIGURE 4. $\mathbb{Z}_3$ counter

a 16 state DFA can have a semigroup with 16^{16} elements in the worst case; understanding the group composition of a semigroup of that size would be very difficult. Fortunately, Krohn and Rhodes proved a deep theorem characterizing the structure of Transformation Semigroups and in turn DFA which shows that semigroups admit a decomposition into smaller groups and semigroups very similar to the Jordan Holder Decomposition [4]. First, I give the full result in the language of Semigroups and then we look at the Holonomy Decomposition [8] a variant of the theorem that has been computationally implemented [10] and interpreted in the language of DFA.

Theorem 1. *(Krohn-Rhodes) Let (S, X) be a finite Transformation Semigroup then there exist $(S_1, X_1), \ldots, (S_n, X_n)$ such that $(S_n, X_n) \wr \cdots \wr (S_1, X_1)$ is homomorphic to (S, X) where S_i is either a Simple Group or the Flip-Flop Semigroup for $1 \leq i \leq n$*

This result was very surprising since although semigroups do not have as much structure as groups, they admit a decomposition very similar to the Jordan-Holder Decomposition for groups, which decomposes groups into simple groups that cannot be further reduced. The product in the Krohn-Rhodes decomposition is not a direct product, but a wreath product. The components of a direct product are independent whereas in a wreath product there is hierachical dependence between the coordinates. This dependence will be made explicit as in the language of DFAs as a cascade product.

Definition 4. *(Cascade Product) Let $A_1 = (\Sigma, Q_1, \delta_1)$ be an automaton and let $B_2 = (Q_1 \times \Sigma, Q_2, \delta_2)$ be an automaton such that for every $q_1 \in Q_1$ and $q_2 \in Q_2$, either $\delta_2(q_2, (q_1, \sigma))$ is defined for every $\sigma \in \Sigma$ or it is undefined for every σ. The cascade product $A_1 \circ A_2$ is the automaton $C = (\Sigma, P, \bar{\delta})$ where $Q = \{(q_1, q_2) : \delta_2(q_2, \langle q_1, \sigma \rangle)$ is defined $\}$ and $\bar{\delta}(\langle q_1, q_2 \rangle, \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \langle q_1, \sigma \rangle))$. The cascade product of more than two automata is defined as $A_1 \circ A_2 \cdots \circ A_k = (\ldots((A_1 \circ A_2) \circ A_3 \ldots) \circ A_k$*

Figure 5 shows two machines that contain $\mathbb{Z}_4$ in their semigroups. but the second is a cascade product of two machines that contain $\mathbb{Z}_2$ in their transformation semigroups. Both machines can count to four, but note that the second machine counts in binary. The top level machine in the cascade denotes whether the number is even or odd and the bottom

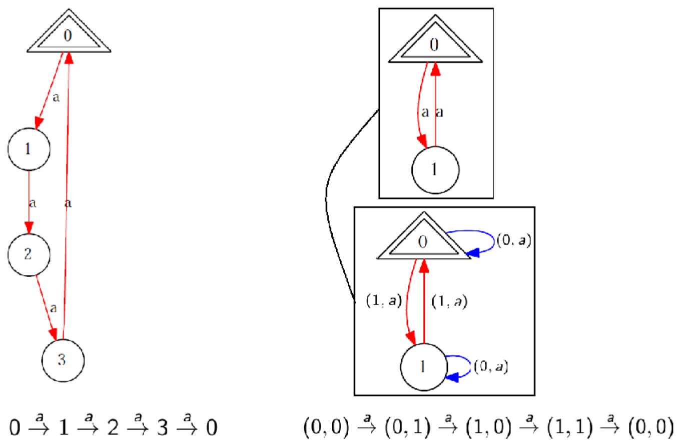
0
1
2
3
a
(0, a)
(1, a)
$0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 3 \xrightarrow{a} 0$
$(0,0) \xrightarrow{a} (0,1) \xrightarrow{a} (1,0) \xrightarrow{a} (1,1) \xrightarrow{a} (0,0)$

FIGURE 5. Both machines contain a $\mathbb{Z}_4$ in their transformation semigroup, but the one on the right is a cascade product of two machines

specifies the number further. This is an example of the hierarchical dependence of the coordinates in the cascade or wreath product.

In the language of automata, the Krohn-Rhodes Theorem is given as the Holonomy Decomposition

Theorem 2. *(Holonomy Decomposition) Let (S, A) be a finite Transformation Semigroup then there exist $(S_1, A_1), \ldots, (S_n, A_n)$ $(S_n, A_n) \wr \cdots \wr (S_1, A_1)$ is homomorphic (surjective) to (S, A) where (S_i, A_i) is a permutation-reset transformation semigroups for $1 \leq i \leq n$*

The cascaded decomposition given by the Holonomy Decomposition coordinatizes the behavior of the original machine in a hierarchical manner. The higher levels represent the coarse-grained behavior of the original machine; Figure 5 is an example of this type of coarse-graining where the top level machines keeps track of whether the number is even or odd, a higher level description. Resets in higher level machines restrict possible transitions at lower levels. The number of levels a machine has is the possible ways it could restrict its transition structure given resets at higher levels.

We now are in a position to apply these concepts of complexity and computation, the number of levels and groups in the holonomy decomposition, to understand the evolution of computation in agents playing a game.

4. METHODS

We implemented an evolutionary simulation of repeated prisoner's dilemma similar to [6] where the players are modeled as automata. The definition of an automaton is extended to include an output function $\lambda : Q \to \Sigma$ that defines an output for each state in the state set. Figure 5 explains how two automata play games. Each round of the game individuals play the move given by the output of their current state and take their opponents move as an input and transition to a new state.

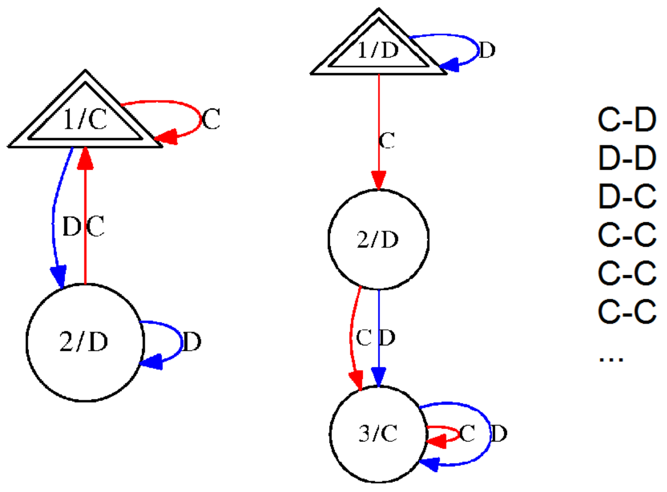

Figure 6. Automata playing games

In our simulation, we used a population of 40 automata and in each generation each player played every other player in 150 rounds of prisoners dilemma in which players were awarded 3 points each for mutual cooperation, 1 point each for mutual defection, and when one player cooperated with the other defecting, the defecting machine received 5 points and the cooperating machine none. The population was initialized with 40 random 16 state automata. At the end of each generation of play, the next generation of players was formed using tournament selection similar to [6] with a 1/3 chance of mutation of state output variable or transition for each player moving on to the next generation and the simulation was run for 60 generations.

5. Results

The dynamics of the system can be characterized by two phases. Initially, since the players are randomly generated, cooperation is difficult to coordinate, and the game is dominated by mutual defection, the Nash equilibrium in this game. However, the machines are eventually able to perform some kind of handshake to recognize their opponents are willing to cooperate and mutual cooperation arises in the population. The first measure we can consider to understand the complexity of the machines is the their number of minimized states. The Myhill-Nerode Theorem implies that for every DFA there exists a minimal machine that is homomorphic to the original [7]. We compared the number of states in the minimal machines of a random population of automata with a population from the evolved cooperative tail of the simulation. In addition to the number of minimized states of a machine, we also considered the number of levels and groups in its Holonomy

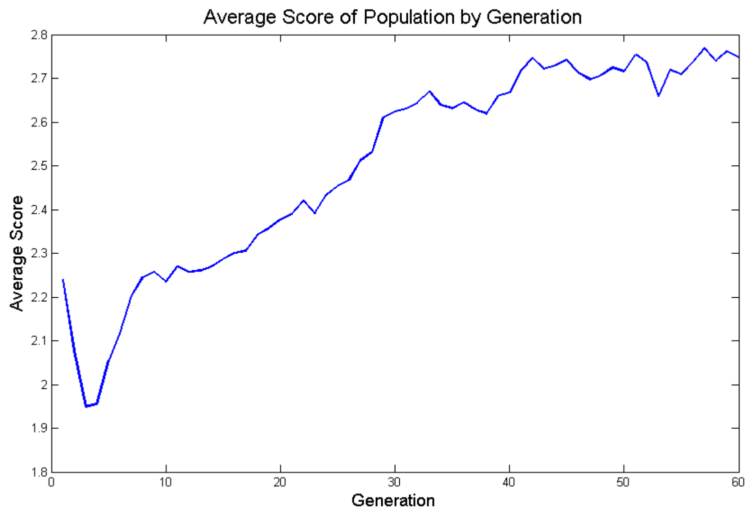

FIGURE 7. Score dynamics averaged over 20 runs of the simulations

Decomposition. For these last two measures, we used Attila's computational implementation of the holonomy decomposition in the sgpdec package of the GAP programming language [11] [12].

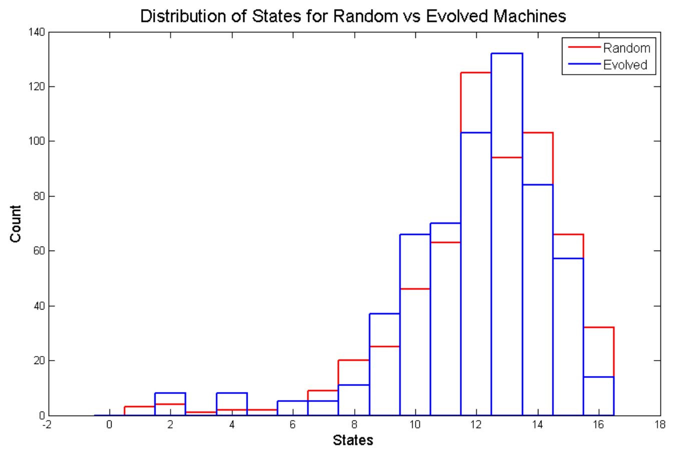

FIGURE 8. Distributions of States in Minimized Machines for Evolved vs. Random Machines

There is no significant difference between the state distributions in this case, which is surprising given that the evolved machines are purportedly just mutually cooperating. We see a similar comparison for groups and levels in the holonomy decomposition.

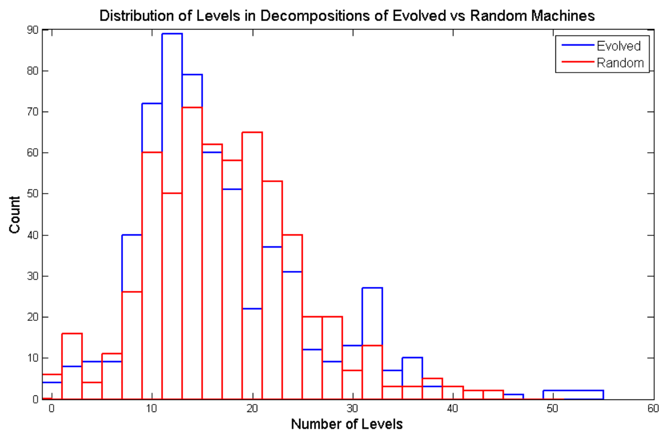

FIGURE 9. Distributions of Levels in Minimized Machines for Evolved vs. Random Machines

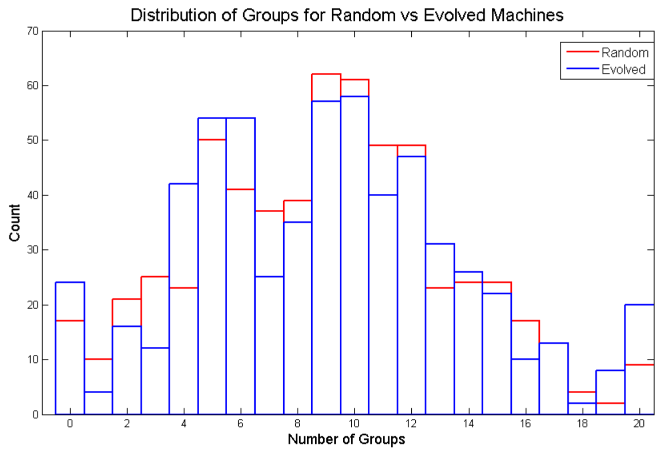

FIGURE 10. Distributions of Groups in Minimized Machines for Evolved vs. Random Machines

All of these measures indicate that the evolved machines are computationally as complex as random machines. There are two explanations for their indistinguishability. First, it is possible the extra states, groups and levels are necessary for the evolved machines to strategically handle a world where mutations can break the norm of mutual cooperation at any point. Second, the pressures of evolution in the simulation may not be strong enough to trim extra states that are accessible but unused in practice, and these extra states can generate spurious groups and levels.

5.1. **Interaction Machine.** To better understand the dynamics of game play and evolution, we define an interaction machine for any two machines that describes the interaction between them.

Definition 5. *Suppose we have two Finite State Automata $A = (Q_A, \Sigma_A, \delta_A, \lambda_A)$ and $B = (Q_B, \Sigma_B, \delta_B, \lambda_B)$ with starting states a_0 and b_0 respectively. Then we define the Interaction Machine of A and B as the Finite State Automaton $I_{A,B} = (\{\tau\}, Q_A \times Q_B, \delta)$ where the alphabet is the singleton $\{\tau\}$ and the transition function δ is defined as follows:*

$$\delta(a_i, b_i, \tau) = (\delta_A(a_i, \lambda_B(b_i)), \delta_B(b_i, \lambda_A(a_i))) = (a_{i+1}, b_{i+1})$$

If we repeatedly apply τ in this fashion starting in state (a_0, b_0), then because $|Q_A|, |Q_B| < \infty$, there exist x and ϵ such that $(a_x, b_x) = (a_{x+\epsilon}, b_{x+\epsilon})$ i.e. eventually the machine enters a loop.

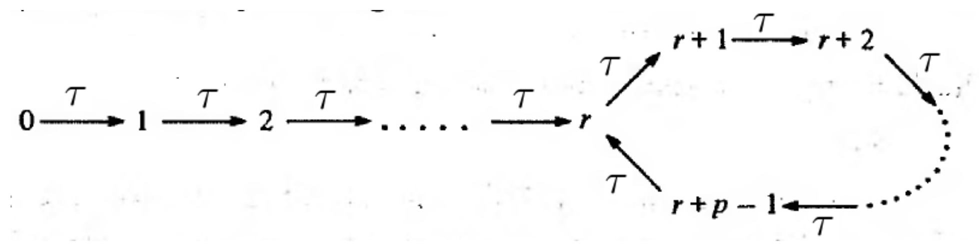

FIGURE 11. Interaction Machine with tail length of r and loop length of p

Since the tail is relatively short with respect to the total number of rounds played in the simulation, the length of the loop can be used to understand the length or complexity of interaction between two machines.

In the evolved limit of the simulation, the average length of the loop is limited to 2 implying that the extra states are excess states that have not been trimmed by the pressures of evolution.

5.2. **Reporting Noise.** To increase the evolutionary pressure on the automata in the hopes of shedding extra states, we introduced reporting noise. Machines use their opponent's move from the previous round as input with which they transition to their next state. Reporting noise is the misreporting of previous moves; this noise exposes more of the automata structure during play. We performed the simulation with two populations of 40 random 12 state machines that competed with machines in the other population and were selected based on intra-population performance, and the the simulation was run for 5000 generations. A two population model was used in this case because eventually it will allow us to implement more complex asymmetric games.

The added noise makes the evolution of cooperation more difficult and more fragile as shown in figure 13. We defined epochs of cooperation and defection as periods of generations where 85% of all interactions between machines in the generation are mutual cooperation or mutual defection respectively. The following histograms present the comparison of states

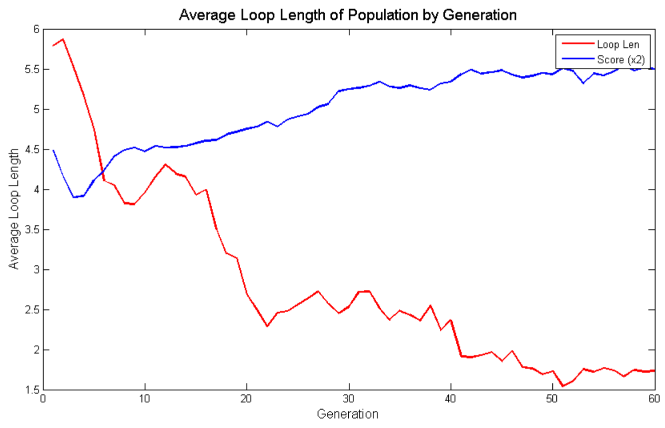

Figure 12. Average Loop Length of a population machines taken as the average over the average of pair-wise interaction loop lengths of machines in a population

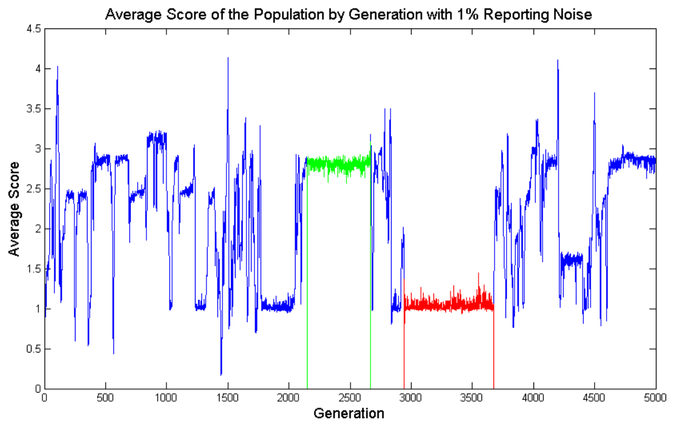

Figure 13. Green indicates a Cooperative Epoch and Red a Defection Epoch

groups and levels for Random vs. Cooperating or Defecting machines defined by these epochs.

Figure 14 reveals that defecting machines are very simple, one state machines that repeatedly defect. Machines with one state have trivial TS and zero levels in their holonomy decomposition, so those measures are degenerate in this case as well and will not be presented.

For cooperating machines, the distribution of minimized states is peaked at 4, smaller than for the random machines. The noise then is removing excess states.

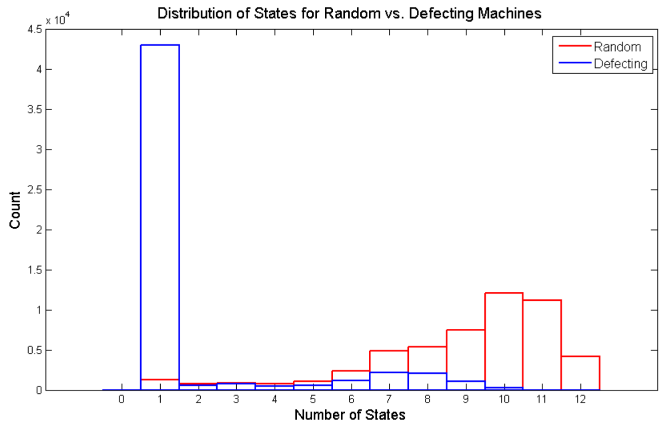

FIGURE 14. Distribution of States for Random vs Defection Machines

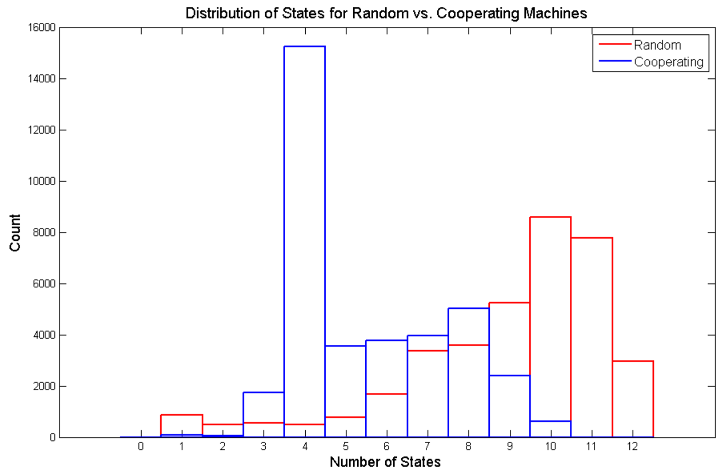

FIGURE 15. Distribution of States for Random vs Cooperating Machines

The number of levels in the holonomy decompositions of cooperating machines does not appear to be very different from random machines, but the distribution for the number of groups is much more peaked at 2 a lower value than for the distribution of the random machines. Generally the cooperating machines have the same number of levels but fewer groups in their holonomy decomposition than random machines implying that there are more resets in the decompositions of the cooperating machines. Thus, the cooperating machines in this simulation are more quickly able to recognize their opponent and play the relevant strategy.

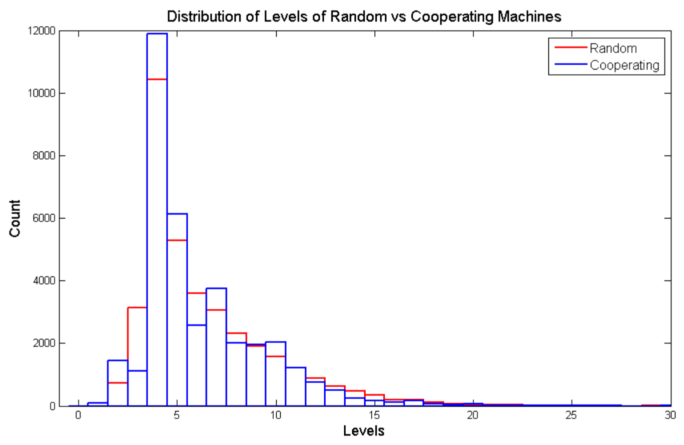

Figure 16. Distribution of Levels for Random vs Cooperating Machines

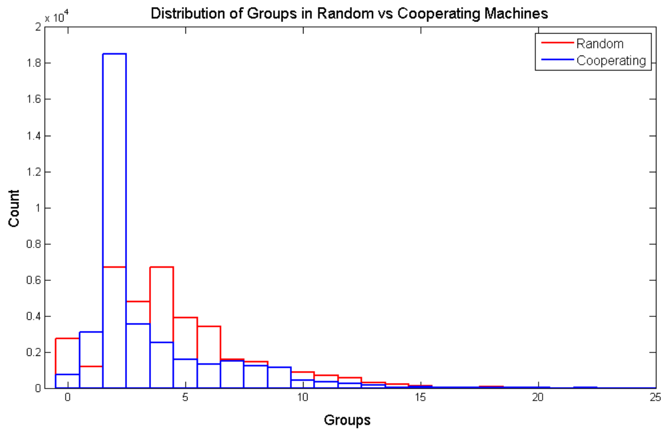

Figure 17. Distribution of Groups for Random vs Cooperating Machines

6. conclusion

The relative simplicity of the machines evolved to play repeated prisoner's dilemma reflects the simplicity of the task. Further work could simulate more complex games to see if this affects the computation in evolved machines. initially though, further work must be done to better understand the nature and function of computation in the machines, which is tied to the composition of the environment automata are adapting to. For games, the environment is the population a machine exists within, which can be heterogenous and difficult to characterize. To better understand the connection between computational structure and function, we should consider problems that pose a more defined and constant environment for the automata to adapt to. Additionally, we can explore the groups associated with the loop in the interaction machine to distinguish between the computations that are possible in a machine and the computations that are executed in practice. Also, the hierarchical nature of the decomposition is relatively unexplored in this paper. It would be interesting in the future to see if learning or evolution occurs hierarchically, if the machines learn or evolve functions at higher coarse-grained levels of the decomposition first.

References

[1] F. Isaacs J. Hasty, D. McMillen and J. J. Collins. Computational studies on gene regulatory networks:in numero molecular biology. *Nature REv. Gen.*, (2):268–279, 2001.

[2] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325:69–110, 2004.

[3] Ion C. Baianu and Updated Hsiao Chen Lin. Copyright ic baianu. computer models and automata theory in biology and medicine: Computer simulation and computability of biological systems, 1985.

[4] J.L. Rhodes and C.L. Nehaniv. *Applications of automata theory and algebra: via the mathematical theory of complexity to biology, physics, psychology, philosophy, and games*. World Scientific, 2010.

[5] Attila Egri-Nagy and Chrystopher L. Nehaniv. Hierarchical coordinate systems for understanding complexity and its evolution, with applications to genetic regulatory networks. *Artificial Life*, 14(3):299–312, 2008.

[6] John H. Miller. The coevolution of automata in the repeated prisoner's dilemma. *Journal of Economic Behavior & Organization*, 29(1):87–112, January 1996.

[7] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.

[8] S. Eilenberg. *Automata, languages, and machines*. Number v. 2 in Automata, Languages, and Machines. Academic Press, 1976.

[9] Oded Maler. On the krohn-rhodes cascaded decomposition theorem. In *Essays in Memory of Amir Pnueli*, pages 260–278, 2010.

[10] Attila Egri-Nagy and Chrystopher L. Nehaniv. Algebraic hierarchical decomposition of finite state automata: Comparison of implementations for krohn-rhodes theory. In *CIAA*, pages 315–316, 2004.

[11] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4.12*, 2008.

[12] Attila Egri-Nagy and Chrystopher L. Nehaniv. *SgpDec – Hierarchical Composition and Decomposition of Permutation Groups and Transformation Semigroups*, 2008.