The Impact of Cellular Representation on Finite State Agents for Prisoner's Dilemma

Daniel Ashlock
Department of Mathematics and Statistics
University of Guelph, 50 Stone Road East
Guelph, Ontario, N1G 2R4
dashlock@uoguelph.ca

Eun-Youn Kim
Department of Mathematics
Iowa State University
Ames, Iowa, 50010
eunykim@iastate.edu

ABSTRACT

The iterated prisoner's dilemma is a widely used computational model of cooperation and conflict. Many studies report emergent cooperation in populations of agents trained to play prisoner's dilemma with an evolutionary algorithm. Cellular representation is the practice of evolving a set of instructions for constructing a desired structure. This paper presents a cellular encoding for finite state automata and specializes it to play the iterated prisoner's dilemma. The impact on the character and behavior of finite state agents that results from using the cellular representation is investigated. For the cellular representation presented a statistically significant drop in the level of cooperation is found. Other differences in the character of the automaton generated with a direct and cellular representation are reported. This paper forms part of an ongoing study of the impact of representation on evolved agents for playing prisoner's dilemma.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Miscellaneous

General Terms

Design

Keywords

Cellular Representation, Prisoner's Dilemma, Evolutionary Computation

1. INTRODUCTION

The Prisoner's Dilemma [4, 3] is a classic model of cooperation and conflict in game theory. Two agents each decide, without communication, whether to cooperate (C) or defect(D). The agents receive individual payoffs depending on the actions taken. The payoffs used in this study are shown in Figure 1. In the Iterated Prisoner's Dilemma (IPD) the agents play many rounds of the Prisoner's Dilemma. IPD is widely used to demonstrate emergent cooperative behaviors in populations of selfishly acting agents [7, 8, 13] and is often used to model biological systems [15], including sociology [12], psychology [14], and economics [11]. It has been tacitly assumed in many publications that the representation of the IPD is inconsequential to the outcome of the simulation.

		Player 1: C	Player 1: D
Player 2	C	(3,3)	(5,0)
	D	(0,5)	(1,1)

Figure 1: Score matrix used for the prisoner's dilemma in this study. Scores are given for (Player 1,Player 2).

This paper introduces a cellular representation for finite state automata. These automaton are used to play the iterated prisoner's dilemma. A *cellular representation* stores directions for building a structure rather than directly storing the byte-level specification for (or parameters of) the structure. Cellular representation, for artificial neural nets, was introduced by Frederic Gruau [9, 10]. Cellular encodings are analogous to incorporating a form of developmental biology into the representation. Transforming the DNA code for an organism into the complete organism requires a complex developmental biology in nature. The cellular encoding undergoes a similar but far simpler process, reading a set of directions and using them to construct, in stages, the final form of the digital organism.

The remainder of the paper is structured as follows. In Section 2 we specify the finite state automaton used together with their direct and cellular representations. In Section 3 we give the design of the experiments and specify the data collected. In Section 4 we present results. Section 5 presents conclusions and places the results in the context of an earlier study that found substantial variation in cooperation across other representations [1].

GECCO'05, June 25–29, 2005, Washington, DC, USA.

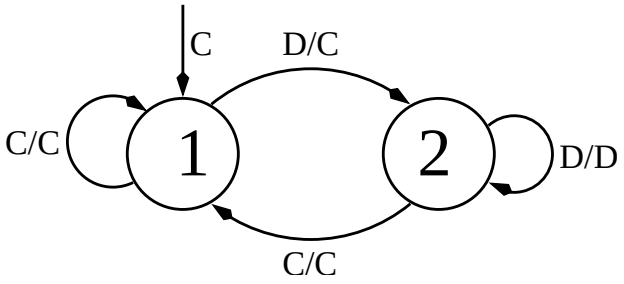


Figure 2: An example of a finite state automaton for playing the iterated prisoner's dilemma. This machine is an implementation of tit-for-two-tats the defects only if its opponent has defected on the last two actions.

2. DIRECT AND CELLULAR REPRESENTATIONS

The finite state automaton used here are Mealy machines augmented with an initial action. These automaton are described in some detail in [2]. When a pair of finite state automata are used to play the iterated prisoner's dilemma the initial actions are used to initiate play. Thereafter the two automata each use the other machine's last action as their current input, generating their action from their finite state transitions. An example of the type of machine encoded by both the direct and cellular representations is given in Figure 2. The initial action is the label on the single sourceless arrow. Transitions are given by arrows and labeled with input/output pairs.

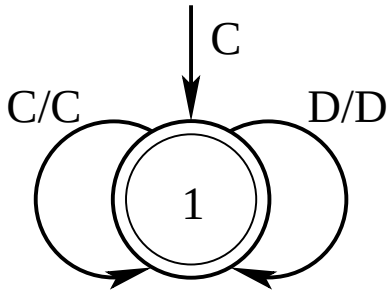
The direct representation uses the following chromosome or data structure. It stores finite state automata as a pair of integer variables giving the initial action and initial state together with an array of states. Each state contains the four integers describing the next state and responding action for transitions out of that state in response to an input of defection or cooperation. The variation operators for the direct representation are as follows. Crossover performs a two-point crossover of the array of states, associating the initial state and action with the first state. The mutation operator changes one of the integers in the machine, replacing it with a valid value selected uniformly at random. This integer is the initial state or action 5% of the time each, a transition 40% of the time, or an action 50% of the time. The usage of these operators and other algorithm parameters are given in the experimental design.

The cellular representation for finite state automata modifies an initial one state machine with a series of editing commands, using a string of editing commands as the chromosome. The initial machine has an initial action the action with the smallest numerical index in the coding scheme (cooperation in this case). The initial machine returns its input as its next output, echoing the other player's actions. We call this machine an *echo machine*. For IPD the echo machines is an encoding of the famous tit-for-tat strategy.

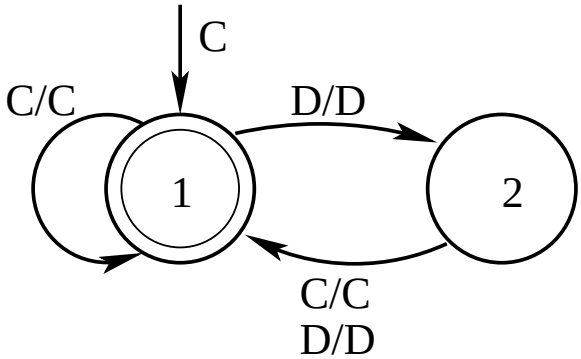
The rules for modifying the initial echo machine into the machine encoded by the cellular representation are given in Table 1. In order to execute these rules it is necessary to use structures not in the final finite state automata. The first of these structures is the *current state pointer*. This pointer is initialized to point to the echo automaton's sole state and designates which state to which to apply an editing command. This pointer may be moved and it may drag a transition arrow with it. This is done by using the *pin* command to associate one of the transition arrows out of the current state with the current state pointer. Later the *release* command places the head of the "pinned" transition arrow on the current state as designated by the position of the current state pointer. While pinned the transition arrow's head moves with the current state pointer and is implicitly released if still pinned at the end of the contruction of an automaton. The second structure beyond the nominal finite state automata is a collection of pointers connecting each state in the machine to the state that was duplicated to create it. These pointers are used to execute the A(ncestor) command. Example 1 gives and example of the construction of a machine implementing the strategy Pavlov from an echo (tit-for-tat) machine.

EXAMPLE 1. *Let's look at the results of starting with Echo (Tit-for-Tat in the Prisoner's Dilemma) and applying the following sequence of editing commands:* D_1, M_1, P_0, F_1, F_0, *or, if we issue the commands using the inputs and outputs of the Prisoner's Dilemma:* D_D, M_D, P_C, F_D. F_C. *The current state is denoted by a double circle on the state.*

Tit-for-Tat is the starting point.



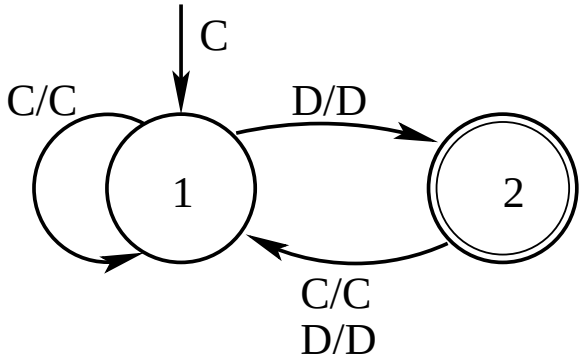
D_D inserts a copy of 1 as the new destination of 1's D-transition.



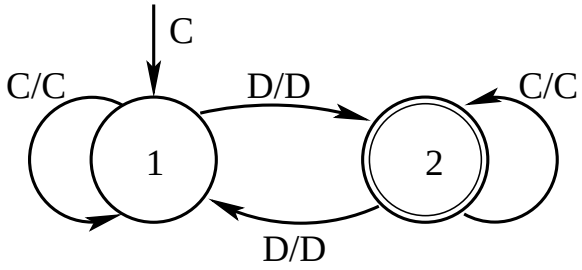
M_D moves the active state to state 2.

Command	Effect
B (Begin)	Increment the initial action modulo the number of actions.
F_n (Flip)	Increment the response associated with the transition for input n out of the current state modulo the number of actions.
M_n (Move)	Move the current state pointer to the destination of the transition for input n out of the current state.
D_n(Duplicate)	Create a new state that duplicates the current state as the new destination of the transition for input n out of the current state.
P_n (Pin)	Pin the transition arrow from the current state for input n to the current state. It will move with the current state until another pin command is executed.
R (Release)	Release the pinned transition arrow, if there is one.
I_n(Square)	Move the transition for input n out of the current state to point to the state you would reach if you made two transitions associated with n from the current state.
A(Ancestor)	Move the current state to the state that was duplicated to create the current state or do not move it if the current state is the initial state.

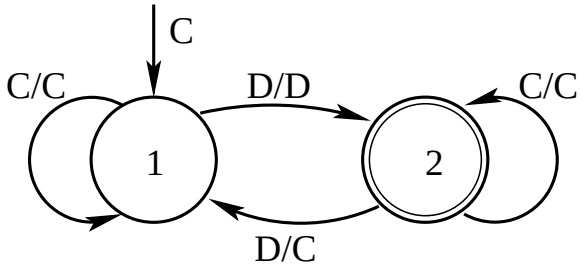
Table 1: Cellular rules for creating finite state automata. For IPD n takes on the possible values Cooperate and Defect. Incrementing an action is done modulo the number of actions and so simply exchanges C and D.



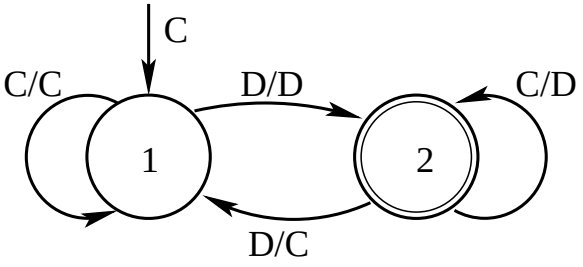
P_C pins the C-transition from the current state to the current state.



F_D increments the response on the D-transition from the current state.



F_C increments the response on the C-transition from the current state.



So, this sequence of editing commands transforms our starting automaton, Tit-for-Tat, into a version of Pavlov.

Notice that only two of the thirteen rules, D_C and D_D generate a new state. This means the number of states in a automaton is equal to the number of D_n rules in the cellular encoding plus one.

The variation operators for the cellular representation are quite simple. The crossover is two point crossover of the string of editing rules. Mutation consists of choosing a position in the string of editing rules uniformly at random and replacing the character at that position with a new one selected uniformly at random. Details of how these operators are applied are given in the section on experimental design.

2.1 Completeness

The baseline among the experiments presented here are those using the direct representation. Here we demonstrate that the automaton that can be represented by the direct representation can all be found in the cellular representation.

Definition 1. *Accessible states in a finite state automaton are the states that can be reached from the initial state.*

The behavior of a finite state automaton only depends on its accessible states. We don't have to consider about inaccessible states when we prove the cellular representation is complete.

Definition 2. *The depth of the state i is the number arcs in the shortest path from the initial state 1 to the state i. It is denoted as $depth(i)$. A depth of a finite state automaton is maximum of all depths of states of G and is denoted as $depth(G)$.*

Notice a state with depth k cannot have an incoming arc from a state with depth less than $k-1$, otherwise it's depth should be less than k and it contradicts that its depth is k. Therefore a state with depth k can have incoming arc only from a state with depth $k-1$. For the same reason, a state with depth k cannot have an outgoing arc to a state with depth more than $k+1$.

THEOREM 1. *Every finite state automaton (ignoring states inaccessible from the initial state) is specified by cellular rules in Table 1. In other words, the given cellular representation in Table1 is complete.*

PROOF. we just need to consider making states and changing transition arrows since we can set the responses using F_n when the state is created without any problem.

Let V be a set of states and E a set of transitions of a finite machine. Then a finite state automaton can be considered as a digraph G with a set of vertices V and a set of arcs E. It is enough to show that a digraph G can be specified by the cellular rules and it can be proved by the mathematical induction on the depth of a finite state automaton G.

If the depth of a finite state automaton is 0, G has only one state which is the initial state. Just change it's action using F_n as appropriate, and we are done.

Suppose this *Theorem* holds when the depth of a finite state automaton is less than k. Consider the case that the depth of a finite state automaton is k. Let V_{k-1} is the set of states whose depth are $k-1$ and V_k the set of states whose depth are k. Delete all states in V_k from G and change the incoming arcs to V_k to loops i.e. let G'=G-V_k+{(i,i)| (i,j) is arc of G such that i in V_{k-1} and j in V_k}. Clearly, each state of G' has two outgoing arcs and it is connected from the initial state so it is a finite states automaton. By the way that it is constructed, its depth is $k-1$. Thus it is specified using the cellular encoding rules by the induction hypothesis. For each outgoing arc (i,j) from V_{k-1} to V_k in G, there is a loop (i,i) in G'. Create the state j which duplicates the state i of G' and move the current state from i to j. Then we have an arc (i,j). Pin and release its two transitions then it become an echo machine.

If the state j has a transition which reaches to any state h in $V-V_k$, then pin that transition of the current state j. Move the current state form j to i using A as needed. The rule A allows the current state in anywhere to get back to the initial state 1 and the rule M_n allows the current state to move from the initial state to any state. In other words, the editing pointer can get from the current state to every other state. It is possible to make states inaccessible from the initial state but they are not part of the finite state automaton and so we may ignore them. In other words, the editing focus can get from the current state to every other state using A and M_n. Thus we can move the current state from the state j to the state h. Release the pinned transition arrow then we have an arc (j,h). If the state j has a transition which reaches a state l in V_k and l is not created yet, then leave that transition as a loop (j,j). Move the current state to the state l' in V_{k-1} which is connected to l. Create the state l by the same way that we created the state j. Move the current state from l to j using A and M_n. Now, we have state l. Pin the loop (j,j) and move the current state from j to l then we will have an arc (j,l). If l is already created then omit the process for creating and do the same process as above.

If another state g in V_{k-1} has an outgoing arc to the state j in the component, it should corresponds a loop (g,g) in G'. Thus move the current state to g and pin to that loop and move the current state back to j using A and M_n.

If there are more states in depth k that should be created then move current state to a state in depth $k-1$ which has an arc to them using M_n and A and repeat same process as above until we create all states and arcs. Therefore the finite state automaton G is specified by the cellular rules. □

3. EXPERIMENTAL DESIGN

The experimental design follows that of [2] and [13] in many details to facilitate comparison across experiments. Two sets of 400 evolutionary simulations were performed, one for the direct representation and on for the cellular representation. Each simulation was run for 250 generations with a population of 36 IPD agents. To evaluate fitness a round robin tournament in which each pair of agents play 150 rounds of IPD was used. Fitness was normalized to average payoff per play for reporting purposes.

The model of evolution used is a generational evolutionary algorithm with a two-thirds elite. In each generation the most fit two thirds of the population was copied into the next generation. These are the *elite*. In sorting the population to find the elite ties are broken uniformly at random. The remaining one-third of the new population was generated as follows. Pairs of distinct parents were selected with replacement by roulette selection from the elite. These parents were copied, the copies subjected to crossover, and a single mutation applied to each result of the crossover to generate pairs of members of the new population.

The automata used in the direct representation have 16 states for consistence with previous experiments[1, 2, 13]. The values for initial states and action and all outputs and transitions were initialized by filling in valid values chosen uniformly at random. In order to make the automata as similar as possible across the two representations the length of the cellular representation was chosen to be 98 editing commands. Two of the thirteen commands create a new state and so, on average, executing 97.5 editing commands will yield a 16 state machine. As we will see in the discussion section this normalization was probably not that important.

The statistics tracked during evolution were the population mean, standard deviation, and maximum of both fitness and age for each evolutionary simulation performed. The age of an automaton is initially zero and in incremented each time the automaton is copied into a new generation as part of the elite. The age of automata can be used to track succession. When a new type arises and takes over an ecology then the maximum and average age drop.

4. RESULTS

It was found that the direct and cellular representations produce statistically significant differences in the level of cooperation. Figure 3 shows a plot of 95% confidence interval for the population mean fitness computer over all 400 evolutionary simulations run for the two representations. While both exhibit the standard dip-and-rise the cellular representation moves to its long-term average value far faster.

In [1] the probabilities that a given representation would be cooperative or at least achieve a better score than a random player flipping a fair coin to choose its move were com-

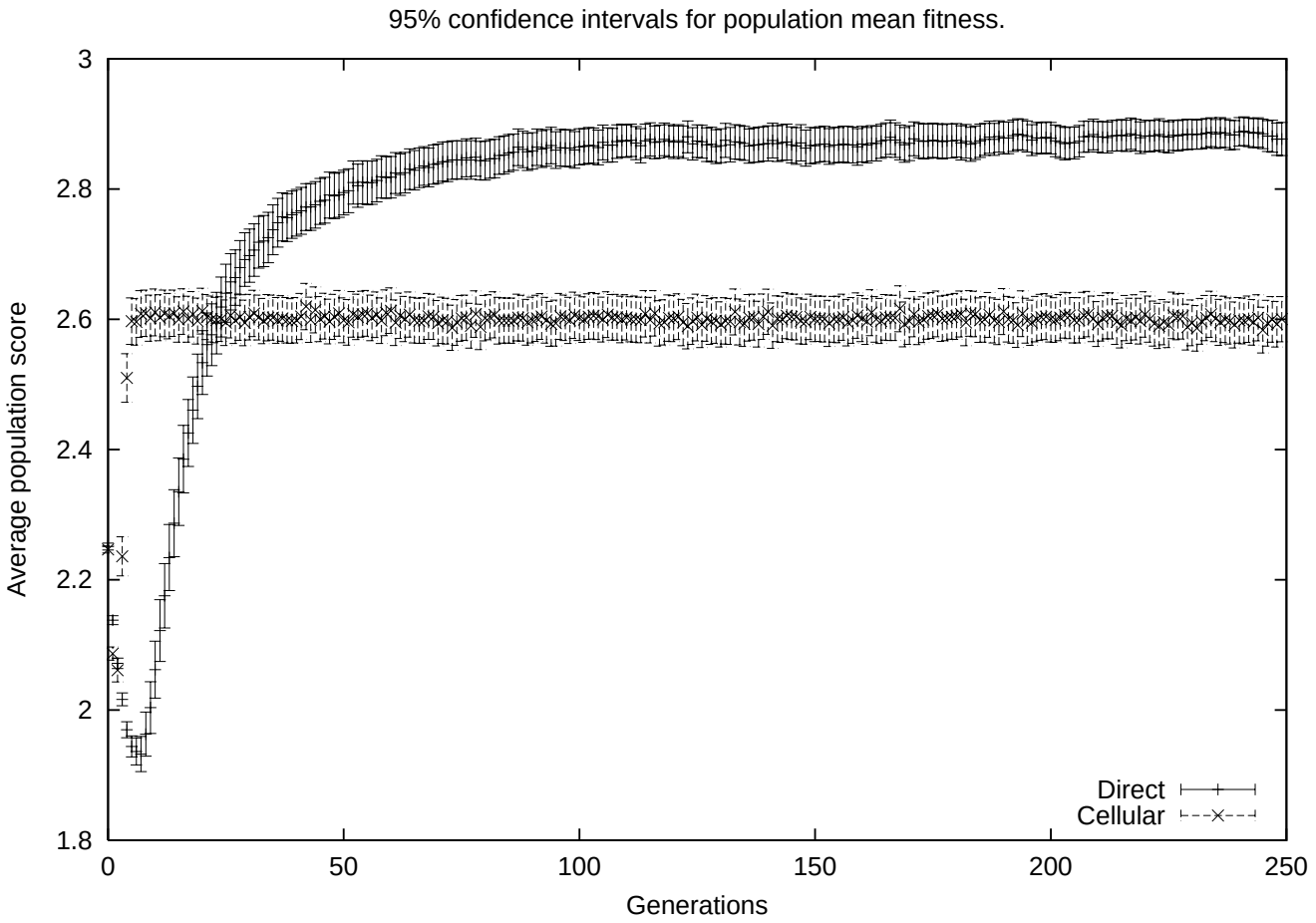


Figure 3: The 95% confidence intervals, computed over 400 populations, for the population mean fitness in generations 1-250.

puted. A population is deemed *cooperative* if its population mean score per game played is at least 2.8. The reasoning used to select this cutoff is given in [1]. Briefly, a sixteen state machine cannot engage in sustained defection as part of its cyclic behavior and achieve a score above 2.8 when fitness is evaluated with 150 rounds of play. A 95% confidence interval was computed for the probability of cooperative and better-than-random play for eight representations: directly represented finite state automata with 16 states, boolean logic trees coded with genetic programming, a Markov chain representation, a lookup table, an alternative genetic programming structure called an ISAc list, boolean logic trees augmented with a time delay operation, and two types of artificial neural net, one with a bias toward cooperation. Details of these representations are given in [1].

The confidence intervals on cooperative and better-than-random play were computed for 100 evolution runs for all representations *except* the direct and cellular representations for finite state automata. For these two representations the 400 runs performed for this paper were used. Because of this the confidence interval is roughly half as wide for these two representations. The confidence intervals for all nine representations are shown in Figures 4 and 5.

The direct and cellular representations exhibit a signifi-

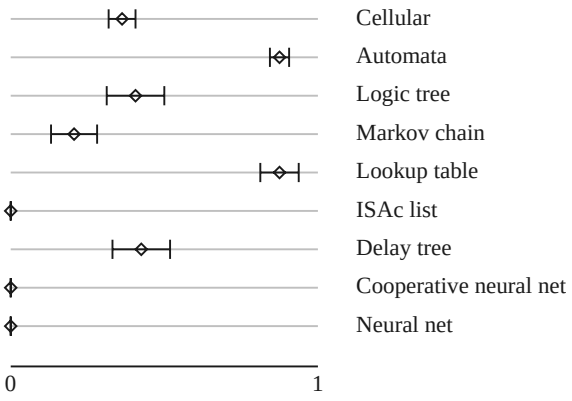


Figure 4: The 95% confidence intervals on the probability a given representation will be cooperative in generation 250.

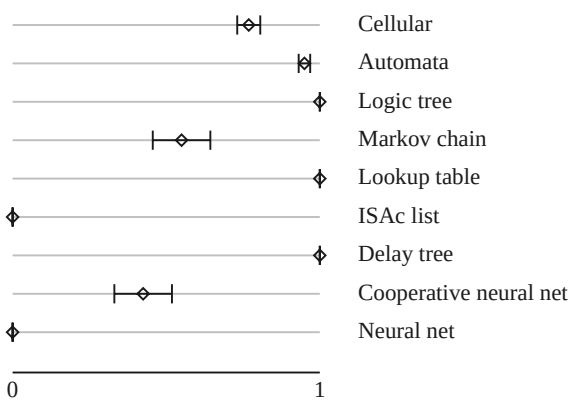


Figure 5: The 95% confidence intervals on the probability a given representation will score better than random in generation 250.

cant difference for both probability of cooperation and probability of better-than-random play. Compared with the other representations the Cellular representation exhibits an intermediate level of cooperativeness and roughly a 75% chance of playing better than random. Notice that by considering the confidence intervals for better-than-random play we see that the cellular representations has behavior significantly different from all the other representations presented.

5. CONCLUSIONS AND DISCUSSION

This paper tests a cellular representation for finite state automata in the context of evolving agents to play the iterated prisoner's dilemma. Three measures of performance, mean population score, probability of cooperative play, and probability of better-than-random play all clearly separate the cellular representation from the direct representation. These performance measures are neither independent nor completely dependent on one another. They all speak to the question "is cooperation emerging in this system?"

Comparison with a previous study of eight relatively simple representations showed that the cellular representation was in the middle of the pack from the perspective of agents evolving to a state of cooperative play. The more cellular representation, which was more complicated from an implementation perspective, was both different from its own direct representation and roughly as cooperative as two other representations. These representations were both forms of genetic programming using parse trees with Boolean operators.

The experiments demonstrate that completeness of representation, proved for the cellular encoding in Section 2.1, does not tell the entire story. Changing the representation modifies the distribution of initial strategies and the shape of the fitness landscape. For the system studied here these impacts were significant.

The complexity of a strategy coded by a finite state automaton can be indexed by the number of states in the machine. For the direct representation this is exactly sixteen states. For the cellular representation, the initial population starts with a shifted binomial distribution of numbers of states with $n = 98$ and $p = \frac{2}{13}$ and a shift caused by starting with one state in the initial machine. This yields a mean of $2 \times \frac{2}{13} + 1 = 16.08$. The standard deviation of the number of states in the cellular representation is $\sqrt{n \times p \times (1-p)} = 3.57$. The experimental design intentionally made the mean number of states in the cellular machine as close as possible to the value for the direct representation.

The number of states, however, is a weak measure of complexity. When a representation does not need to use all of its data then some of the space may be filled in with junk such as the bloat encountered in genetic programming [5]. This junk space can be used to manage the disruption of the variation operators and to store things that may be useful after being revealed by a fortuitous crossover. In a finite state automaton, states that cannot be reached from the initial state are one possible type of junk. Figure 6 shows the distribution of the number of states accessible from the initial state in 100,000 randomly generated machines for both the direct and cellular representation. The distributions are almost mirror images with the cellular representation favoring enormously less complex initial automata. The computations used to generate Figure 6 were performed again on the 400 final populations of 36 automata. This yields 14,400 rather than 100,000 samples. The resulting distribution of numbers of states of shown in Figure 7.

Comparison of Figure 6 and 7 suggests that the cellular representation is locating simpler automata than the direct representation. Aside from weeding out machines with a very small number of accessible states from the cellular populations there seems to be little change in the distribution, given the smaller size of the evolved samples.

Prior to performing the research the authors thought that the cellular representation would lead to machines with a reasonably large number of accessible states. Each state is accessible, when created by a *duplicate* operation, from the state it duplicates. The *pin* operation permits the breaking of these ties and apparently has a larger impact than expected.

There are other measures of complexity that "states accessible from the initial state" for a finite state automata. If two states produce exactly the same responses for all possible input strings then those states are *equivalent* and the machine can be simplified by identifying those states. The simplification does not change the strategy encoded by the finite state automata, it just reduces the number of states required to implement it. Checking all input strings of a given length $2^n - 1$ suffices to document equivalence of states because a finite state automata must fall into a cycle for strings of inputs exceeding that length (in fact far more efficient algorithms are possible, but that is beyond the scope of this paper). The finite state machines obtained for both the standard and cellular representations were first reduced by throwing out all states not accessible from the initial state and then reduced by identifying all sets of equivalent states. Table 2 gives a 95% confidence interval for the mean number of states in these doubly reduced machines for both the standard and cellular representations.

This last quantitation of the automata complexity computed for each representation demonstrate that the two representations are sampling the strategy space in substantially different ways.

6. FUTURE DIRECTIONS

The cellular representation introduced in this paper is one

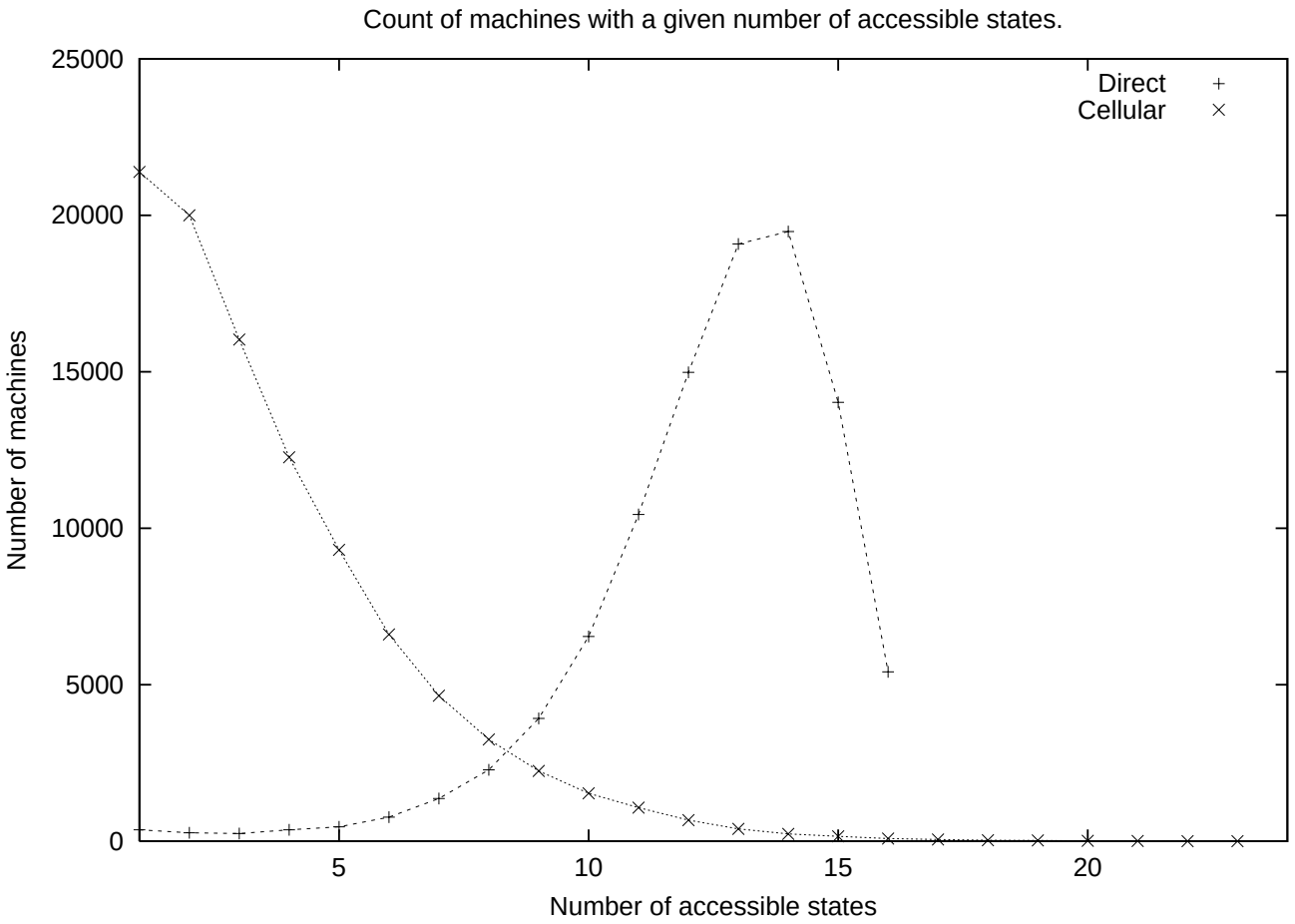


Figure 6: Counts of the number of automata with a given number of states accessible from the initial state. This plot is for a sample of 100,000 randomly generated automaton for each of the two representations.

Representation	Mean states	95% Confidence Interval
Standard	10.75	(10.69,10.81)
Cellular	3.54	(3.51,3.57)

Table 2: Mean automata sizes for both representations after reduction of inaccessible and inequivalent states.

of many possible cellular representations. New editing operations could be added, old one deleted, or the rate of use of editing operations could be changed. Reducing or eliminating the *pin* operations, for example, might remove the potentially confounding problem with the automata having few accessible states.

This paper is part of an ongoing project to catalog representations and understand their impact on the iterated prisoner's dilemma and other games. The goal of modeling behavior for use in everything from virtual ecology to digital focus groups waits on an understanding of how representation impacts system behavior. Only after substantial progress has been made in this area can we make agents that behave in a manner usefully similar to the animals and people we wish to model. As the project continues Figures 4 and 5 will continue to grow and will be generalized to other games.

An important next step is the comparison of data on human or animal behavior, e.g. [6], to the behavior of various representations. The project that this study is a part of seeks to explore a rich enough space of representations that human (or guppy) like representations are not difficult to locate when designing a sociological or ethnological simulation.

7. ACKNOWLEDGMENTS

The authors would like to thank David Fogel for contributing thoughts on the role of granularity in representational sensitivity and for many stimulating discussions on evolution of agents playing the iterated prisoner's dilemma. The authors also gratefully acknowledge the support of the Department of Mathematics and Statistics of the University of Guelph in the matter of funding and of space for the second author.

8. REFERENCES

[1] D. Ashlock and N. Leahy. A representational study of

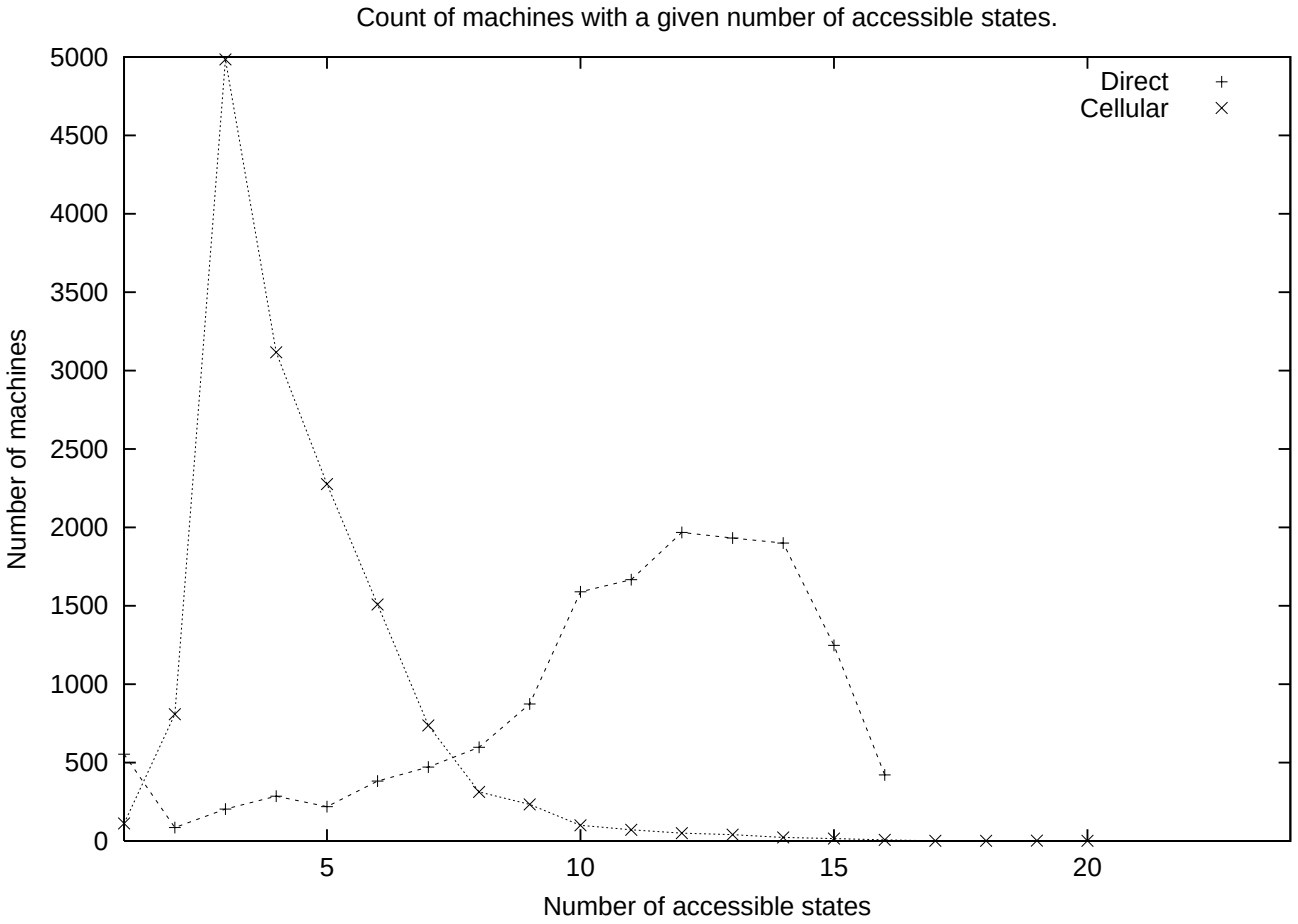


Figure 7: Counts of the number of automata with a given number of states accessible from the initial state. This plot is for generations 250 of the experimental runs for each representation.

game theoretic simulations. In *SMCia/03: Proceedings of the 2003 IEEE Conference on Soft Computing in Industrial Applications*, pages 67–72, Piscataway NJ, 2003. IEEE Press.

[2] D. Ashlock, M. D. Smucker, E. A. Stanley, and L. Tesfatsion. Preferential partner selection in an evolutionary study of prisoner's dilemma. *Biosystems*, 37:99–125, 1996.

[3] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[4] R. Axelrod and W. D. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, 1981.

[5] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming : An Introduction : On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, San Francisco, 1998.

[6] L. A. Dugatkin and M. Mesterton-Gibbons. Cooperation among unrealted individuals: Reciprocal altruism, byproduct mutalism and group selection in fishes. *Biosystems*, 37:19–30, 1996.

[7] D. Fogel. Evolving behaviors in the iterated prisoners dilemma. *Evolutionary Computation*, 1(1), 1993.

[8] D. B. Fogel. On the relationship between the duration of an encounter and the evolution of cooperation in the iterated prisoner's dilemma. Working Paper, July 1994.

[9] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, France, 1994.

[10] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2):151–183, 1995.

[11] M. Hemesath. Cooperate or defect? russian and american students in a prisoner's dilemma. *Comparative Economics Studies*, 176:83–93, 1994.

[12] J. M. Houston, J. Kinnie, B. Lupo, C. Terry, and S. S. Ho. Competitiveness and conflict behavior in simulation of a social dilemma. *Psychological Reports*, 86:1219–1225, 2000.

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

[14] D. Roy. Learning and the theory of games. *Journal of Theoretical Biology*, 204:409–414, 2000.

[15] K. Sigmund and M. A. Nowak. Evolutionary game theory. *Current Biology*, 9(14):R503–505, 1999.