Fingerprinting: Visualization and Automatic Analysis of Prisoner's Dilemma Strategies

Daniel Ashlock
Mathematics and Statistics
University of Guelph,
Guelph, Ontario
Canada N1G 2W1
danwell@iastate.edu

Eun-Youn Kim
National Institute For Mathematical Sciences
385-16, Doryong-dong,
Yuseong-gu, DaeJeon 305-340,
South Korea
eunykim@nims.re.kr

***Abstract*— Fingerprinting is a technique for generating a representation-independent functional signature for a game playing agent. Fingerprints can be used to compare agents across representations in an automatic fashion. The theory of fingerprints is developed for software agents that play the iterated prisoner's dilemma. Examples of the technique for computing fingerprints are given. The paper summarizes past results and introduces the following new results. Fingerprints of prisoner's dilemma strategies that are represented as finite state machines must be rational functions. An example of a strategy that does not have a finite state representation and which does not have a rational fingerprint function is given: the majority strategy. It is shown that the AllD- and AllC-based fingerprints can be derived from the tit-for-tat fingerprint by a simple substitution. Fingerprints for four new probe strategies are introduced, generalizing previous work in which tit-for-tat is the sole probe strategy. A trial comparison is made of evolved prisoner's dilemma strategies across three representations: finite state machines, feed forward neural nets, and lookup tables. Fingerprinting demonstrates that all three representations sample the strategy space in a radically different manner, even though the neural net's and lookup table's parameters are alternate encodings of the same strategy space. This space of strategies is also a subset of those encoded by the finite state representation. Shortcomings of the fingerprint technique are outlined, with illustrative examples, and possible paths to overcome these shortcomings are given.**

I. Introduction

Evolutionary game theory can use an evolutionary algorithm to generate a vast number of different game playing agents in a short time. It is typically impractical to figure out *which* game playing agents were evolved for a number of reasons. Many representations for evolvable game playing agents, e.g. neural nets or finite state automata, are capable of producing thousands of different encodings of the same strategy. In addition, evolved game playing agents are often cryptic and complex. The effort required for direct analysis of evolved structures varies from the time-consuming to the impractical. It is possible to create one-to-one representations that encode a small number of possible strategies, both solving the problem of understanding the evolved agents and depriving evolution of much of its scope to produce interesting strategies.

This study presents *fingerprinting*, a representation-and-encoding independent method of identifying game playing agents. The method is developed for the iterated prisoner's dilemma and applied to three representations: finite state machines, lookup tables, and artificial neural nets. The theory of fingerprints is developed to the point of supporting a rapid sampling-based algorithm for approximating fingerprints. This permits the rapid identification, sorting, and classification of game playing agents. While this study deals only with prisoner's dilemma, the technique extends to any simultaneous two-player game with a finite number of moves. Software for working with fingerprints is available from the first author on request.

The prisoner's dilemma [14], [13] is a classic model 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. The payoff for mutual cooperation C is the *cooperation* payoff. The payoff for mutual defection D is the *defection* payoff. The two asymmetric action payoffs S and T, are the *sucker* and *temptation* payoffs. In order for a two player simultaneous game to be considered the prisoner's dilemma, it must be the case that

$$S < D < C < T \tag{1}$$

and

$$2C > (S + T). \tag{2}$$

The first inequality simply puts the payoffs in their intuitive order. The second insists that alternating cooperation and defection in a pair of players (taking turns stabbing one another in the back) pay off no better on average than does mutual cooperation.

		$\mathcal{S}$	
		C	D
$\mathcal{P}$	C	3	5
	D	0	1

(1)

		$\mathcal{S}$	
		C	D
$\mathcal{P}$	C	C	T
	D	S	D

(2)

Fig. 1. (1)The payoff matrix for prisoner's dilemma used in this study – scores are earned by strategy $\mathcal{S}$ based on its actions and those of its opponent $\mathcal{P}$ (2) A payoff matrix of the general two player game – $C, T, S,$ and D are scores given for the game as well.

In the *iterated prisoner's dilemma* (IPD) the agents play many rounds of the prisoner's dilemma. IPD is widely used to model emergent cooperative behaviors in populations of selfishly acting agents and is often used to model systems in biology [28], sociology [20], psychology [27], and economics [19]. Many different sorts of evolutionary computation systems have been used to evolve agents to play the iterated prisoner's dilemma. In [17] particle swarm optimization is used to co-evolve agents. Threaded finite state machines that permit multiple action threads based on the agent's internal state are used in [4]. A form of Cartesian genetic programming, called function stacks, are used to encode agents in [2]. Permitting prisoner's dilemma agents to evolve within a spatial framework [22], [3] has an impact on both the chance that cooperation will arise and on which strategies evolve. Other version of the prisoner's dilemma are also studies with evolutionary computation. In [16] the authors implement multiple levels of cooperation and defection and include noise.

This study gathers together and extends the underlying theory of *fingerprints* for game-playing agents and applies fingerprints to a study of the types of prisoner's dilemma strategies that arise under evolution for different representations. Finite state machines, feed-forward neural nets, and lookup tables are found to sample the space of strategies in very different ways.

Fingerprints are developed in detail in a pair of theses [30], [23]. A portion of the theory of fingerprints and an initial application to the visualization of evolved agents appears in [6]. Additional applications as well as a marriage of fingerprints with a new technique called *multi-clustering* appear in [7]. Multiclustering, a technique for clustering that avoids artifacts induced by choice of distance measure, is defined and explored in [8]. The theory of fingerprinting is both summarized and extended in this study. Among the new results presented are a substantial extension of the number of known fingerprints, a generalization of fingerprints to more probe strategies than tit-for-tat, and a proof that the fingerprint of a finite state machine is always a rational function (previously they were known to be power series). An algorithm for accurately approximating the fingerprints of finite state machines too large for the computation of exact fingerprints is also given.

Fingerprinting was used in [12], with a finite state representation, to demonstrate that the strategies that arise have different distributions for different population sizes and in different epochs. The latter result, that strategies rare or absent at the beginning of evolution become common after thousands of generations of evolution, was surprising. In [9] fingerprints were used to demonstrate that the rate of appearance of several well-known strategies varied between a direct finite state representation for prisoner's dilemma playing agents, a cellular representation for finite state agents, and a new type of representation called a *function stack*, a modified form of Cartesian Genetic Programming [24].

The study in [9], continued in [5], investigates the effect of changing the representation used for a prisoner's dilemma agent. The representations covered by the two studies are two versions of feed forward neural nets (one biased at the neuron level toward cooperation), Boolean parse trees [15], with and without a one-step time delay operation, a linear genetic programming representation called an ISAc list [1], lookup tables, a type of Markov chain [26], and both a direct and cellular [5] representation of finite state machines. The change of representation, with other factors held as near to constant as possible, yielded a change from 0% to 95% in the probability that final populations were cooperative. Using fingerprints to examine the sets of strategies that arise puts a finer point on these distinctions.

The remainder of the study is structured as follows. In Section II the theory of fingerprints is summarized and new results are presented with proofs given in an appendix. The design of experiments performed are given in Section III. Results and conclusions are presented in Section IV. Possible future directions for both the improvement of fingerprinting techniques and applications are given in Section V.

II. Theoretical Results

This section develops the theory of fingerprints. Results proved elsewhere have their proofs cited, new results have their proofs presented in an appendix. We start with a brief list of known strategies in Table I. All of these strategies, except random and majority, can be realized as finite state machines. An example of a finite state machine of the kind used in this study, a Mealy machine, is given in Figure 2.

Ripoff		
Initial response: D		
Initial state: 1		
State	If D	If C
1	$C \rightarrow 3$	$C \rightarrow 2$
2	$C \rightarrow 3$	$D \rightarrow 1$
3	$D \rightarrow 3$	$C \rightarrow 3$

Fig. 2. A finite state implementation of the strategy *Ripoff*.

The play of two finite state machines in the presence of noise can be represented as a *Markov process*. This allows the determination of an expected (average) score for any pair of strategies by standard techniques in stochastic processes [26]. We will use game playing agents with strategies that incorporate parameterized noise to fingerprint other agents. The strategy used to evaluate other agents is called the *probe* strategy. The fingerprints have independent variables that establish the character of the noise and return a dependent variable that is the expected score of the agent being fingerprinted against the probe strategy. Noise represents probabilities of cooperating or defecting in spite of the move the probe strategy would normally have made. The fingerprint will thus be a map from probabilities, (x, y), of "irrational" (non-probe strategy) cooperation and defection, respectively, to a value, E, the expected score against the noisy agent.

Definition 1: If $\mathcal{A}$ is a strategy for playing the iterated prisoner's dilemma, then $JA(\mathcal{A}, x, y)$ (*Joss-Ann of* $\mathcal{A}$) is a

TABLE I

EXAMPLES OF PRISONER'S DILEMMA STRATEGIES.

Always Cooperate(AllC) This strategy always plays C.

Always Defect(AllD) This strategy always plays D.

Fortress-3(Fort3) This strategy is an example of a strategy that uses a password. If the opponent defects twice in a row (the password) and cooperates thereafter, then Fortress-3 will cooperate. Any deviation from this sequence resets the need to defect twice. A minimal finite state implementation of Fortress-3 is shown in Figure 3. Fortress-3 was first defined in [12] and is an example of a strategy that only arises after substantial evolution has taken place.

Majority(Maj) This strategy returns a play equal to the majority of its opponent's plays, breaking ties in favor of cooperation. Majority has no finite state representation.

Pavlov(Pav) The strategy, Pavlov, plays C as its initial action and cooperates thereafter if its action and its opponent's actions matched last time. A minimal finite state implementation of Pavlov is shown in Figure 3.

Periodic CD(PerCD) This strategy cooperates and defects on alternate moves no matter what its opponent does.

Psycho(Psy) The strategy, Psycho, chooses D as its initial action and then plays the opposite of its opponent's last action.

Random(Rand) The Random strategy simply flips a fair coin to decide how to play. Random has no finite state representation.

Ripoff(Rip) This strategy alternates cooperation and defection until its opponent defects for the first time. On the round after this defection, it cooperates and then plays tit-for-tat thereafter.

Thumper(Thmpr) This strategy cooperates initially. If its opponent defects, then it defects on the next two moves; if its opponent's second move after defection is cooperate, it continues cooperating; otherwise it defects twice as before. A minimal finite state implementation of Thumper is shown in Figure 3.

Tit-for-tat(TFT) The strategy, tit-for-tat, plays C as its initial action and then repeats the other player's last action.

Tit-for-two-tats(TF2T) This strategy defects only if its opponent has defected on the last two moves.

Tit-for-three-tats(TF3T) This strategy defects only if its opponent has defected on the last three moves.

Two-tits-for-tat(TTFT) This strategy defects on the two moves after its opponent defects, otherwise it cooperates.

strategy which has a probability x of choosing the move C, a probability y of choosing the move D, and otherwise uses the response appropriate to the strategy $\mathcal{A}$.

If S is the space of strategies for playing the iterated prisoner's dilemma, then the Joss-Anne modification of strategies can be viewed as a function $JA : S \times F \mapsto S$ where $F = \{(x, y)|x, y \in R, 0 \leq x+y \leq 1\}$ that yields a continuum of strategies. The notation JA comes from the initials for Joss and Ann. Joss was a player submitted to Axelrod's famous computer tournament for the iterated prisoner's dilemma. It would occasionally defect without provocation in hopes of a slight improvement in score. Ann is the first name of A. Stanley who suggested the addition of random cooperation [29], [10] instead of random defection. When $x + y = 1$, the strategy $\mathcal{A}$ is not used, and the resulting behavior is a random strategy with play probabilities (x, y). In more general terms, a JA strategy is an alteration of a strategy, $\mathcal{A}$, that causes the strategy to be played with random noise inserted into the responses. When playing, the strategy, $\mathcal{A}$, also updates its own

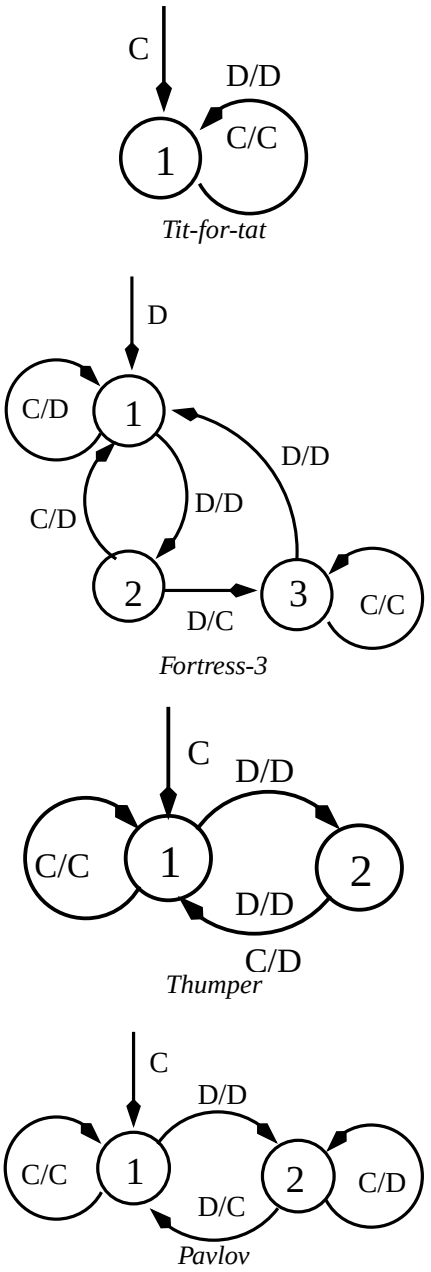


Fig. 3. Minimal finite state implementations of the prisoner's dilemma strategies Tit-for-tat, Fortress-3, Thumper, and Pavlov.

internal state. It does this even when an action generated at random is used instead of its own action. Nesting the Joss-Ann construction yields no new strategies.

Lemma 1: The strategy $JA(JA(\mathcal{A}, x_1, y_2), x_2, y_2)$ is equivalent to $JA(\mathcal{A}, x_2+(1-x_2-y_2)x_1, y_2+(1-x_2-y_2)y_1)$.

Proof: [6], Lemma 1 ■

Definition 2: A **fingerprint** $F_{\mathcal{A}}(\mathcal{S}, x, y)$ with $0 \leq x, y \leq 1$, $x + y \leq 1$ for strategy $\mathcal{S}$ with probe $\mathcal{A}$, is the function that returns the expected score of strategy $\mathcal{S}$ against $JA(\mathcal{A}, x, y)$ for each possible (x, y). The double fingerprint $F_{\mathcal{AB}}(\mathcal{S}, x, y)$ with $0 \leq x, y \leq 1$ returns the expected score of strategy $\mathcal{S}$ against $JA(\mathcal{A}, x, y)$ if $x + y \leq 1$ and $JA(\mathcal{B}, 1 - y, 1 - x)$ if

$x + y \geq 1$. In this case $\mathcal{A}$ is the **lower probe** and $\mathcal{B}$ is the **upper probe**.

While the fingerprint function itself is often possible to find, it is the graph or the approximation of the graph that is often used in analysis. This is in part because a useful approximation of the graph of the function can be computed in cases where the analysis to find the actual function would be intractable. The concept of the double fingerprint was introduced to extend the fingerprint to the unit square in a natural fashion. A unit square is preferable because it is more easily manipulated by a computer, is more easily viewed by humans, and it uses paper more efficiently.

To compute and understand fingerprints, $Markov\ chains$ are required; familiarity at the level given in [26] is assumed in the remainder of this paper. A focused review of the theory appears in [23].

Example Fingerprint Computation

The fingerprint of the strategy Pavlov, using tit-for-tat as a probe, is computed as an example in this section. A minimal finite state representation of Pavlov is shown in Figure 3. To find a fingerprint, the first step is to construct a $Markov\ chain$ for the two strategies involved. In the case of $F_{tit-for-tat}(Pavlov, x, y)$, the set of ordered pairs

$$\{(C1, D1),(C1, C1),\ (D2, D1),\ (D2, C1)\}$$

forms the (accessible) state space. A letter denotes an action. The numbers in a pair denote the internal (finite state machine) states of strategies Pavlov and tit-for-tat, respectively. Readers should verify for themselves that the given combinations of internal states and actions cover all attainable possibilities. Then, constructing the transition matrix, P, for the Markov chain is just a matter of putting the transition probabilities between the states in matrix form.

	$(C1, D1)$	$(C1, C1)$	$(D2, D1)$	$(D2, C1)$
$(C1, D1)$	0	0	y	$1-y$
$(C1, C1)$	y	$1-y$	0	0
$(D2, D1)$	$1-x$	x	0	0
$(D2, C1)$	0	0	$1-x$	x

Because this Markov chain consists of one finite communicating class, it has the stationary distribution π (again see [26]), and it can be found by solving the equations $(P'-I)\pi = 0$ and $\sum \pi(i) = 1$. We obtain:

$$\pi = (\frac{y(1-x)}{2y(1-x)+x(1-x)+y(1-y)},$$

$$\frac{x(1-x)}{2y(1-x)+x(1-x)+y(1-y)},$$

$$\frac{y(1-x)}{2y(1-x)+x(1-x)+y(1-y)},$$

$$\frac{y(1-y)}{2y(1-x)+x(1-x)+y(1-y)}).$$

With π in hand, computing the expected score can be completed by taking a dot product of π with the appropriate score vector $(S, C, D, T)'$ which gives corresponding scores for ($(C1, D1)$, $(C1, C1)$, $(D2, D1)$,$(D2, C1)$). This yields the fingerprint function

$$F_{tit-for-tat}(Pavlov, x, y) =$$

$$\frac{Sy(x-1)+Cx(x-1)+Dy(x-1)^2+Ty(y-1)}{2y(x-1)+x(x-1)+y(y-1)}$$

In the case of the iterated prisoner's dilemma, we usually score $S = 0, C = 3, D = 1$,and $T = 5$ so

$$F_{tit-for-tat}(Pavlov, x, y) =$$

$$\frac{3x(x-1)+y(x-1)^2+5y(y-1)}{2y(x-1)+x(x-1)+y(y-1)}$$

A shaded plot of this function appears in Figure 5. The shading is a tool that permits the rapid identification of the fingerprint function in visualizations, as demonstrated in [7]. We note that while the fingerprint as defined so far only exists for non-negative x and y such that $x + y \leq 1$ that the function is meaningful over the entirety of the unit square $0 \leq x, y \leq 1$. This is Theorem 3 in the following section.

A. Characterization of Fingerprint Functions

The lemmas and theorems in this section are quite abstract and may remind the reader of material from an advanced calculus or introductory real analysis class. They are included for two reasons. First of all, as we saw in computing the tit-for-tat fingerprint of Pavlov, fingerprint computation is laborious. In order to compute the fingerprint of a strategy implemented as an n-state FSA with a probe implemented as an m-state FSA, the inversion of a $4nm \times 4nm$ matrix whose entries are bivariate polynomials is required. The theorems in this section demonstrate that the fingerprints of strategies implemented as finite state machines belong to the class of rational functions without singularities or discontinuities in the interior of the fingerprint. This means that a sampled approximation to fingerprints, an algorithm for which is presented in Section II-B, is practical and stable on any set of samples in the interior of the fingerprint.

The second reason for presenting these results is the beauty of the double fingerprint of a strategy and its dual, given in Theorem 3. The noise variables x and y that define the continuum of strategies used to compute a fingerprint are only well-defined in the triangle in which x and y are both positive and sum to at most one. The double fingerprint exploits a fundamental similarity of the fingerprints of a strategy and the transposed fingerprint of its dual to demonstrate that the fingerprint function exists in the entire square $0 \leq x, y \leq 1$. Unlike the rest of the fingerprint theory this beautiful result is unique to games with two moves: the natural analog of the double fingerprint for games with three moves, e.g. rock-paper-scissors, does not exist.

The set $\mathcal{F}$ is the triangle $\{(x, y) : 0 \leq x, y$ and $0 \leq x+y \leq 1\}$. We denote the interior of this triangle by $\mathcal{F}'$.

Theorem 1: A double fingerprint, $F_{\mathcal{AB}}(\mathcal{S}, x, y)$, is continuous at a point if and only if the fingerprints $F_{\mathcal{A}}(\mathcal{S}, x, y)$ and $F_{\mathcal{B}}(\mathcal{S}, x, y)$ are continuous.

Proof: [6], Theorem 1. ■

Theorem 2: If $\mathcal{S}$ and $\mathcal{A}$ are strategies representable by finite state machines, then $F_{\mathcal{A}}(\mathcal{S}, x, y)$ is continuous over $\mathcal{F}'$.

Proof: [6], Theorem 2. ■

Corollary 1: If $\mathcal{S}$ and $\mathcal{A}$ are strategies representable by finite state machines, then $F_{\mathcal{A}}(\mathcal{S}, x, y)$ is infinitely differentiable over the interior of the unit square.

Proof: [6], Corollary 1. ■

Definition 3: Strategy $\mathcal{A}'$ is said to be the **dual** of strategy $\mathcal{A}$ if $\mathcal{A}$ and $\mathcal{A}'$ can be written as finite state machines that are identical except that their responses are reversed.

The strategies tit-for-tat and Psycho are examples of dual strategies. Tit-for-tat repeats its opponent's last choice. Psycho plays the opposite of its opponent's last choice. A strategy can be its own dual. For example, the strategy Pavlov is a self-dual strategy, as shown in Figure 4. Given the same input string, it generates reversed responses if its initial action is reversed.

		Pavlov
	input	$CCDCDDDCDCD...$
$First\ action\ C$	$response$	$CCCDDCDCCDD...$
$First\ action\ D$	$response$	$DDDCCDCDDCC...$

Fig. 4. An example of a self-dual strategy

Theorem 3: If $\mathcal{A}$ and $\mathcal{A}'$ are dual strategies, then $F_{\mathcal{AA}'}(\mathcal{S}, x, y)$ is identical to the function $F_{\mathcal{A}}(\mathcal{S}, x, y)$ extended over the unit square.

Proof: [6], Theorem 3. ■

Corollary 2: If $\mathcal{A}$ and $\mathcal{A}'$ are dual strategies, then $F_{\mathcal{AA}'}(\mathcal{S}, x, y)$ is infinitely differentiable over the interior of the unit square.

Proof: [6], Corollary 2. ■

The dual fingerprint is the most beautiful result in [30]. The fact that the fingerprints obtained using a strategy and its opposite as probes generate translated versions of the same function is startling. That they can be fit together in a single analytic function covering the unit square is unexpected and yields an excellent visualization of fingerprints, an example of which is shown in Figure 5.

Definition 4: A **closed communicating class**(CCC) in a finite state machine is a set of states such that there is a set of transitions from each state to each other in the class and no transitions leaving the class.

Definition 5: We call states of a finite state machine **transient** unless they are members of a closed communicating class.

Since the set of states accessible from a given state in a finite state machine is, by definition, non-empty, it follows that a finite state machine can always be decomposed into its transient states and those that are members of some CCC. The set of transient states may be empty in which case the machine is composed of a single CCC. In [23] it is shown that a prisoner's dilemma strategy implemented as a finite

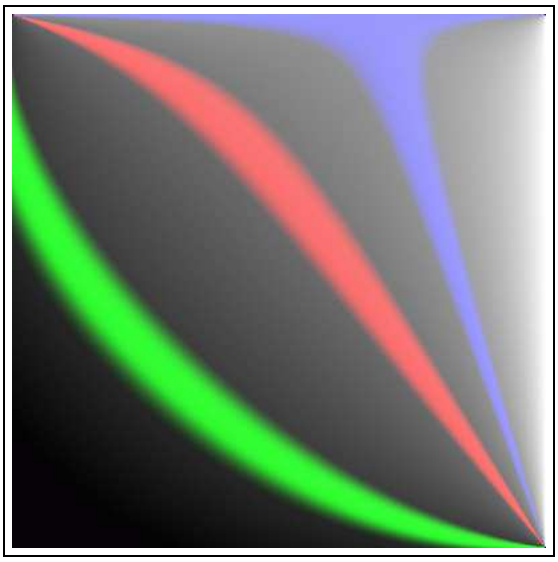


Fig. 5. A shaded plot of $F_{tit-for-tat}(Pavlov, x, y)$ for prisoner's dilemma. Lighter colors represent higher scores with black=0 and white=5. Shading is modified to show three important parts of the score space. The high shaded band represents scores within a narrow range of the score for mutual cooperation. The middle shaded band similarly represents the score obtained by mutual random play. The low shaded band marks scores near the score for mutual defection. The differential shading of this representation provides for rapid visual identification of fingerprints.

state machine has a fingerprint that is a rational function if the state transition diagram of the finite state machine has one CCC. The following theorem demonstrates that the fingerprint function of a strategy realized as a finite state machine is always a rational function. States, within the data structure for a given finite state machine, that are not accessible from the initial state are not considered to be part of the machine in the subsequent material.

Theorem 4: Suppose that S is a strategy for the iterated prisoner's dilemma that is realized as a finite state machine M. The $F_{tit-for-tat}(S, x, y)$ is a rational function of x and y.

Proof: Example 1 can be used as a concrete example that may help the reader in following this proof. It immediately follows this theorem. Select tit-for-tat as the probe strategy. Associate the symbols $F_1, F_2, \ldots, F_n$ with the CCCs of M. The theorem is already true for machines with no transient states, and so assume that M possesses one or more transient states. Note that in this case the initial state must be transient. Construct a tree, called the **F-tree** for M whose root node is the starting state of M. The tree is binary. Nodes in the tree are either (i) transient states of M or (ii) symbols F_i. The daughters of a node N are the transient states that are the destination of a transition out of N or symbols F_i if a transition from N is to a state within a CCC. Symbols F_i are leaves of the tree and so have no daughter nodes – all internal nodes of the tree are transient states of M. A transient state becomes a leaf if another instance of itself appears in a higher level of the tree. The edges of the tree are annotated with the probability that the transition associated with that edge will

occur when playing against Joss-Ann of the probe strategy.

For each internal node N with daughters N_a and N_b write the equation:

$$N = pN_a + qN_b \tag{3}$$

where p is the transition probability from N to N_a, and q is the transition probability from N to N_b. The equations form a set of of k equations in k unknowns where k is the number of internal nodes in the F-tree. Each transient node has an equation because all such nodes are accessible from the initial state. These equations exist in the rational function field over the rational numbers [21]. The left hand sides are the internal nodes of the F-tree, once each, while the right hand sides are either symbols connected with communicating classes or internal nodes of the F-tree multiplied in either case by monomials.

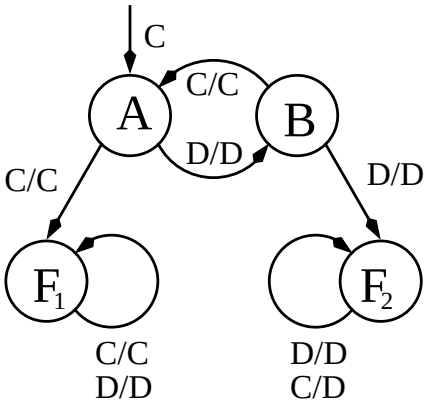
The F-tree can be used to compute the expected score that M gets against the probe strategy, because it contains all possible paths through M together with the probabilities of following them. The expected score starting at the root node R of the tree is the same as the expected score for M, but it is also the sum of the expected value at each of the daughters of R multiplied by their corresponding transition probabilities. A similar relationship holds for all internal nodes in the tree, all the way to the leaves. If a leaf is a symbol F_i, then the expected score at that leaf is the score for the corresponding CCC. If a leaf is a transient state, then a recursion arises that yields the equations described above.

As a result, the solution for R to the system of equations above gives the fingerprint for the strategy encoded by M if we substitute the fingerprint functions for the strategies encoded by the CCCs for the symbols F_i. It is important to know that the fingerprint of a CCC does not depend on the starting state within that class [23].

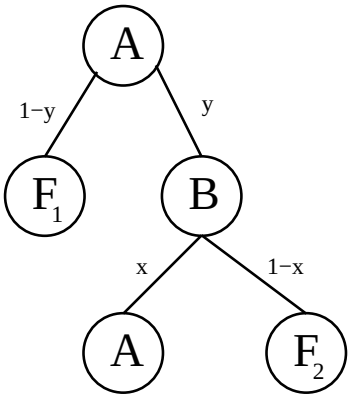
Since the system of equations is over the rational function field, its solution, if it exists, is a rational function. The simple form of the equations can be used to deduce that they do have a solution. Since all terms of the right hand side are multiplied by monomials, it follows that under substitution the degree of the resulting polynomial coefficient of a given symbol on the right hand side is higher degree than its coefficient on the left hand side. This means that we may eliminate a symbol from the right hand side of the equation if it is the left hand side of any equation without degeneracy and then solve the system by simple elimination of variables. We note that the number of equations and unknowns match, and so the system admits a solution and a unique one.

We may conclude that the fingerprint of a prisoner's dilemma strategy using tit-for-tat as a probe is a rational function in x and y. ■

Example 1: Assume that the probe strategy is $tit-for-tat$ and consider the finite state machine M:



This machine has two transient states, A and B, and two communicating classes, F_1 that plays tit-for-tat and F_2 that plays always defect. The F-tree (defined in the proof of Theorem 4) for M is:



The corresponding equations are:

$$A = (1-y)F_1 + yB \tag{4}$$

and

$$B = xA + (1-x)F_2. \tag{5}$$

Solving these equations for A we obtain

$$A = \frac{(1-y)F_1 + y(1-x)F_2}{1-xy}. \tag{6}$$

Following the proof of Theorem 4 we see:

$$F_{tit-for-tat}(M,x,y) =$$

$$\frac{(1-y)F_{tit-for-tat}(tit-for-tat,x,y)}{1-xy} +$$

$$\frac{y(1-x)F_{tit-for-tat}(AllD,x,y)}{1-xy}$$

Theorem 5: For any strategy A for the prisoner's dilemma:

(i) $F_{AllC}(A,x,y) = F_{tit-for-tat}(A,1-y,y)$,

(ii) $F_{AllD}(A,x,y) = F_{tit-for-tat}(A,x,1-x)$.

Proof: Notice that the expected score of A against $JA(AllC,x,y)$ is the same as its score against $JA(tit-for-tat,1-y,y)$ because AllC always cooperates; in other words if $JA(AllC,x,y)$ does not defect because of probability y, then

it cooperates no matter what (with probability $1-y$). Similarly the expected score of A against $JA(AllD, x, y)$ is the same as the expected score against $JA(tit-for-tat, x, 1-x)$. Substitution into the tit-for-tat fingerprints complete the proof. ■

Theorem 5 shows that the AllC and AllD fingerprints contain a subset of the information contained in the tit-for-tat fingerprint.

An Example of a Discontinuous Fingerprint

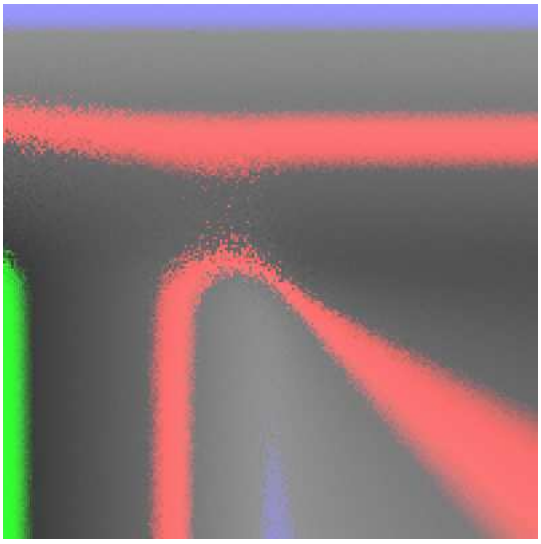


Fig. 6. The fingerprint $F_{tit-for-tat}(Majority, x, y)$, obtained by sampling. The shading scheme is the same as that used in Figure 5.

An example of a strategy that cannot be implemented with a finite state machine and which does not have a rational function fingerprint is the Majority strategy. Since it must keep track of the actual number of cooperations and defections made by an opponent, it requires an unbounded number of states. The fingerprint $F_{tit-for-tat}(Majority, x, y)$ has a discontinuity at the point (0.5,0.5). A sampled representation of this fingerprint is given in Figure 6. The formula for this fingerprint and a proof that it is discontinuous are given in [23], Section 2.1.4. The fingerprint is piecewise rational, splitting the unit square into three regions, with continuity at the boundaries *except* at the point $(0.5, 0.5)$. This example is included to demonstrate that the fingerprints of finite state strategies are a proper subset of the set of fingerprint functions. This has a consequence for use of the algorithm presented in Section II-B: if a representation encodes strategies that cannot be encoded as finite state machines, then the algorithm may be unstable. The grainy character of the center of the picture in Figure 6 results from the central discontinuity. Compare the quality of the images in Figures 6 and 5. Figure 5 shows a strategy that can be implemented with a finite state machine and, as a result, is far smoother.

B. An Algorithm for Approximating Fingerprints

The Markov chain calculations for computing the explicit rational form of the fingerprint function for an n-state finite state machines, using tit-for-tat as the probe, involves the inversion of a $4n \times 4n$ matrix whose entries are monomials x, y, $1-x$, $1-y$. As the number of states in a strategy grows, the resulting computations become quite time consuming. It is possible to rapidly compute accurate values of the fingerprint function for specific values of x and y using the algorithm described below.

If S is the set of states in a finite state machine M, then the set of states of the Markov chain used in fingerprint computation (the Markov state space for M) is $S \times \{CC, CD, DC, DD\}$, representing the actions of $JA(tit-for-tat, x, y)$ and M at each state of M. Depending on the details of M, not all of these states may be used (only half the states were used in the example computation of Pavlov's fingerprint). For each state there are two transitions out to other states with probabilities x and $1-x$ or y and $1-y$. There are two states in the Markov state space (i, IC) and (i, ID) that have non-zero probability on the first move: i is the initial state of M, and I is the initial action of M. For a given set of noise parameters (x, y), the probability of (i, IC) is $1-y$ while the probability of (i, ID) is y – this follows from the fact that the probe strategy, tit-for-tat, cooperates initially. This assignment of positive probabilities to these two states alone is the *initial distribution* of probabilities for the fingerprint Markov chain. Note that each state has a score associated with it for M depending on its second coordinate CC, CD, DC or DD. The initial distribution places a probability of 1 on the Markov state (s, CC) where s is the starting state of the finite state machine.

Algorithm 1: **The Sand Pile Algorithm**
Input: A FSM M with n states and appropriate real values x, y.
Output: $\sim F_{tit-for-tat}(M, x, y)$.
Details:
Construct the Markov state space for M.
Create probability variables for each state.
Load the initial distribution
Repeat $50n$ times
  Update the probability variables for each state S
    The new probability of a variable associated with S
    is the sum over variables associated with states
    that transition to S of the current probability of
    those variables times the probability of transition to S
End Repeat
Sum over the state space((Probability of state)*(score for state))
Return(Sum)

What the sandpile algorithm does is to treat probability like sand. It places the initial probability distribution as two piles of sand on the two states that initially have positive measure. It then moves the sand (probability) among the states according to their transition probabilities. The amount of sand at each state converges rapidly to the asymptotic probabilities for the chain, if they exist. Once these asymptotic probabilities have been computed, the expected score is obtained by adding up the contribution from each state: its probability times the appropriate score. The number $50n$ as the iteration bound

was chosen experimentally as yielding eight decimal places of accuracy on a set of known machines with up to five states.

It is important to note that the asymptotic probabilities need not exist. If the Markov chain is *periodic*[26] then the algorithm can return absurd values. The strategy Periodic CD is one that gives rise to a periodic Markov chain. It is possible to patch the sandpile algorithm to give the correct result as follows. At the point at which the algorithm normally sums the contribution of each state, instead run the algorithm for a large number of additional updatings of the probability variables (1000 in this study) computing the average of the summed contributions over these 1000 steps.

Strategies with periodic Markov chains are very rare (computations not shown), but do arise. For this reason, the variation of the sandpile algorithm that can deal with periodic strategies was used for all approximations in this study. The correctness of the variation of the algorithm was tested on a set of strategies, derived from the periodic strategies described in Section II-C, for which exact fingerprints were computed and compared with the algorithmically approximated fingerprints.

When comparing evolved strategies using fingerprints, a set of 25 values sampled at points (x, y):

$$\left\{\left(\frac{i}{6}, \frac{j}{6}\right) : 1 \leq i, j \leq 5\right\},$$

an equally spaced grid in the interior of the fingerprint, is used. These sampled fingerprints are treated as points in Euclidean 25-space. The standard distance on $\mathbb{R}^{25}$ induces a distance measure on prisoner's dilemma strategies via these sampled fingerprints.

C. Exact Fingerprints

A catalog of known fingerprints using tit-for-tat as the probe, as well as some examples using TF2T or TF3T as the probe, appears in Table II. Recall that Theorem 5 permits us to extract those fingerprints that use AllD and AllC as probes by simple substitution. A number of interesting things can be learned from the table. First of all, notice that tit-for-tat and ripoff have the same fingerprint. This highlights the fact that the fingerprint detects only asymptotic behavior. Against any opponent that ever defects against it, ripoff plays tit-for-tat. Against AllC or TF2T, however, it manages an unanswered defection every other move. Ripoff was discovered in an experiment in which a population of evolving finite state machines was spiked with immortal TF2T; it optimally exploits them. Because any defection against ripoff yields tit-for-tat play, the noise in $JA(\mathcal{A}, x, y)$ causes its play to be asymptotically identical to that of tit-for-tat. The strategies can be distinguished by using sampled fingerprints with a finite number of plays (the authors thank Wendy Ashlock for the suggestion) but this is a topic for the future.

Another unexpected identity is the equal fingerprints of PerCDDC (A strategy that plays CDDC CDDC CDDC ...) and Random when tit-for-tat or TF2T are used as probes. Unlike ripoff and tit-for-tat, these strategies are not even of the same type. PerCDDC is a finite state strategy while random is not. This example motivated the inclusion of fingerprints that use TF2T and TF3T as probes. The two strategies Random and PerCDDC have the same fingerprints when tit-for-tat or tit-for-two-tats are used as probes, and they have distinct fingerprints when TF3T is used as a probe. While the AllC and AllD fingerprints do not change, when the probe changes from tit-for-tat to TF2T, the fingerprints of tit-for-tat and Pavlov do change becoming far more complicated.

In the remainder of this section the fingerprints of an infinite family of periodic strategies are derived.

Definition 6: A *periodic strategy* is one that makes a repeating sequence of moves taken from the set {**C**,**D**}.

The simplest such strategies are AllD and AllC. A periodic strategy is denoted as $Perx_0x_1...x_{k-1}$ where $x_i \in \{C, D\}$. AllD is thus $PerD$ while AllC is $PerC$. The *period* of a strategy $Perx_{01}x_1...x_{k-1}$ is k. A periodic strategy is *reduced* if it is not equivalent to any periodic strategy with a shorter period. $PerDD$, for example is not reduced while $PerCCD$ is.

Theorem 6: The tit-for-tat fingerprint of a periodic strategy $Perx_1x_1...x_k$ is

$$F_{tit-for-tat}(x, y) =$$

$$\frac{N_{CC}(3(1-y)) + N_{CD}(5(1-y)+y)}{k} +$$

$$\frac{N_{DC}3x + N_{DD}(5x + (1-x))}{k}$$

$$= \frac{(3-3y)N_{CC} + (5-4y)N_{CD}}{k} +$$

$$\frac{3xN_{DC} + (4x+1)N_{DD}}{k}$$

where N_{CC} is the number of adjacent cooperates in the circularized periodic string of plays and N_{CD}, N_{DC} and N_{DD} are the similar number of adjacent CD, DC, and DD's in the circularized periodic string of plays respectively.

Proof: The only states in the Markov chain underlying the fingerprint of the periodic strategy are of the form Cx_i and Dx_i. Perform index arithmetic $(mod\ k)$. The Markov transitions out of Cx_i and Dx_i each go to both of Cx_{i+1} and Dx_{i+1}. There are two possibilities. If x_i is C, then the probability of transition from either of Cx_i and D_{xi} to Cx_{i+1} is $1-y$, and the probability of transition from either of Cx_i and D_{xi} to Dx_{i+1} is y. Likewise, if x_i is D, then the probability of transition from either of Cx_i and D_{xi} to Cx_{i+1} is x, and the probability of transition from either of Cx_i and D_{xi} to Dx_{i+1} is $1-x$. From this we deduce that: if x_i is C, then the Markov equations associated with states Cx_{i+1} and Dx_{i+1} are:

$$\begin{aligned} Cx_{i+1} &= (1-y)(Cx_i + Dx_i) \\ Dx_{i+1} &= y(Cx_i + Dx_i), \end{aligned}$$

Tit-for-tat fingerprints

$$F(tit-for-tat,x,y)=\frac{y^2+5xy+3x^2}{(x+y)^2} \qquad F(AllD,x,y)=4x+1$$

$$F(Fort3,x,y)=\frac{3x+7y-6x^2-3xy-5y^2+2x^3+x^2y+5xy^2}{x+4y-2x^2-5xy-y^2+x^3+2x^2y+xy^2} \qquad F(AllC,x,y)=3-3y$$

$$F(Pun1,x,y)=\frac{3x+5y-4y^2}{x+2y} \qquad F(TF2T,x,y)=\frac{3x^2y+5xy-3xy+3x+y^2}{x^2y+2xy^2+x+y^3}$$

$$F(Pav,x,y)=\frac{(3x+y)(x-1)+5y(y-1)}{(x+2y)(x-1)+y(y-1)} \qquad F(Ttit-for-tat,x,y)=\frac{3x^2+5x^2y+4xy+xy^2+y}{x^3+2x^2y+xy^2+y}$$

$$F(Rand,x,y)=\frac{9+7x-7y}{2} \qquad F(Psy,x,y)=\frac{4(y-1)(x-1)+5(y-1)^2}{2(x-1)(y-1)+(x-1)^2+(y-1)^2}$$

$$F(PerCDDC)=\frac{9+7x-7y}{2} \qquad F(PerCD)=\frac{5+3x-4y}{2}$$

$$F(Rip,x,y)=\frac{y^2+5xy+3x^2}{(x+y)^2} \qquad F(Thmpr,x,y)=\frac{12xy+3x^2+2y-4xy^2-4x^2y}{x^2+xy+2y}$$

Tit-for-two-tat fingerprints

$$F(tit-for-tat,x,y)=\frac{(3(1-x+y)(x+x^2y+xy^2-2xy)+5xy+y^2(x+y))}{(y^3+2xy^2+x^2y+x)} \qquad F(AllD,x,y)=4x+1$$

$$F(Pav,x,y)=\frac{(3(1-y)(1-x)(x+y)+5y(1-y)(2-y-x)+y(1-x))}{y^3+2xy^2+-4y^2+x^2y-5xy+5y-x^2+x} \qquad F(AllC,x,y)=3-3y$$

$$F(Rand,x,y)=\frac{6+2x-5y}{2} \qquad F(PerCDDC)=\frac{6+2x-5y}{2}$$

Tit-for-three-tat fingerprints

$$F(Rand,x,y)=\frac{8-7y}{2} \qquad F(PerCDDC)=\frac{7+4x-6y}{8}$$

TABLE II

A TABLE OF KNOWN, EXACT FINGERPRINTS. THE FINGERPRINTS ARE GROUPED BY THE PROBE STRATEGIES USED, PERMITTING THE OMISSION OF THE PROBE STRATEGIES AS SUBSCRIPTS.

while if $xi = D$, then

$$\begin{aligned} Cx_{i+1} &= x(Cx_i + Dx_i) \\ Dx_{i+1} &= (1-x)(Cx_i + Dx_i). \end{aligned}$$

No matter what the value of x_i, adding either of the pairs of equations above yields

$$Cx_i + Dx_i = Cx_{i+1} + Dx_{i+1}$$

from which we may deduce that, for all i, $Cx_i + Dx_i = \frac{1}{k}$. If x_i is C, then

$$Cx_{i+1} = \frac{1-y}{y} Dx_{i+1},$$

while if x_i is D, then

$$Cx_{i+1} = \frac{x}{1-x} Dx_{i+1}.$$

In the former case we obtain

$$Cx_{i+1} = \frac{1-y}{k}$$

and

$$Dx_{i+1} = \frac{y}{k},$$

while in the latter case we obtain

$$Cx_{i+1} = \frac{x}{k}$$

and

$$Dx_{i+1} = \frac{1-x}{k}.$$

Consider the circularized string of periodic plays. The above probabilities permit us to compute the payoff at each state of the Markov chain. For each CC in the string we obtain a score of C with probability $\frac{1-y}{k}$ and S with probability $\frac{y}{k}$. For each CD we obtain a score of T with probability $\frac{1-y}{k}$ and D with probability $\frac{y}{k}$. For each DC we obtain a score of C with probability $\frac{x}{k}$ and S with probability $\frac{1-x}{k}$. Each DD yields a score of T with probability $\frac{x}{k}$ and D with probability $\frac{1-x}{k}$. Adding these up we obtain

$$F_{tit-for-tat}(x,y) =$$

$$\frac{N_{CC}(C(1-y)+Sy)}{k}+$$

$$\frac{N_{CD}(T(1-y)+Dy)}{k}+$$

$$\frac{N_{DC}(Cx+S(1-x))}{k}+$$

$$\frac{N_{DD}(Tx+D(1-x))}{k}.$$

Putting in the usual numerical values we obtain:

$$F_{tit-for-tat}(x,y) =$$

$$\frac{N_{CC}(3(1-y))+N_{CD}(5(1-y)+y)}{k}+$$

$$\frac{N_{DC}3x+N_{DD}(5x+(1-x))}{k}$$

$$=\frac{(3-3y)N_{CC}+(5-4y)N_{CD}}{k}+$$

$$\frac{3xN_{DC}+(4x+1)N_{DD}}{k}.$$

■

In fingerprinting the periodic strategies we obtain an infinite number of fingerprints. It is interesting to note that they are *all* linear functions and in fact are linear combinations of three linear functions: $3-3y$, $4x+1$, and $5+3x-4y$. The first two of these are associates with the CC and DD's in the periodic strategy and are, respectively, the tit-for-tat fingerprints of always cooperate and always defect. The third is the sum of the functions associated with CD and DC. A moment's thought will convince the reader that these always occur in the same numbers – any C before a D is at the beginning of a run of D's that must end in a C and vice versa. The periodic strategies thus have the simplest possible fingerprints. It is also interesting to note that many different periodic strategies have the same fingerprint when tit-for-tat is used as the probe. More complex probe strategies are required to distinguish periodic strategies.

III. Experimental Design

Three representations, finite state machines, lookup tables, and feed forward neural nets, are compared using fingerprinting. For each representation 400 evolutionary runs are performed. The resulting strategies are then compared using fingerprints to see if the representations sample the strategy space in a different manner. Two of the representations, lookup tables and feed forward neural nets, have their parameters (number of inputs) chosen so as to encode identical spaces of strategies. The number of states used in the finite state machines is set to the minimal value that causes the set of strategies encoded by the finite state machines to include that encoded by the other two representations. The lookup tables are indexed by the opponent's last three actions, the neural nets receive these as inputs, while the finite state machines receive the opponent's last action as an input. Since finite state machines are state conditioned, they can "remember" previous actions.

The evolutionary algorithms used in this study operate on a population of 36 agents. Agent quality is assessed by a round-robin tournament in which each pair of players engage in 150 rounds of the iterated prisoner's dilemma. Reproduction is elitist with an elite of the 24 highest scoring strategies. When constructing the elite, ties are broken uniformly at random. Twelve pairs of parents are picked by fitness-proportional selection with replacement on the elite. Parents are copied, and the copies are subjected to variation of the sort defined subsequently for the representation in question. Variation consists of the representation-specific crossover and then the representation-specific point mutation operator, applied once. All the representations permit access to information about three past actions of the opponent. Actions before the start of play are assumed to have been cooperation.

In each simulation, the evolutionary algorithm was run for 250 generations. The elite portion of the final generation was saved for fingerprint analysis. This yields 400 sets of 24 machines or a total of 9600 evolved prisoner's dilemma agents per representation. Many of the parameters used in this study are either selected in imitation of previous studies or are essentially arbitrarily. A careful parameter-variation study would require millions of collections of evolutionary runs because of combinatorial explosion of the parameter space. The current study represents a small slice of this diverse space of possible experiments. We now carefully describe the three representations used together with their variation operators.

Finite State Machines

The finite state machines used in this study are 8-state Mealy machines. State transitions are driven by the opponent's last action. Access to state information permits the machine to condition its play on several of its opponent's previous moves. The machines are stored as linear chromosomes; the first state in the chromosome is the initial state. The binary variation operator used is two-point crossover. Crossover preserves whole states. The point mutation operator changes a single state transition 40% of the time, the initial state 5% of the time,

the initial action 5% of the time, or an action associated with a transition 50% of the time. A mutation replaces the current value with a valid value for the state transition or action is selected selected uniformly at random.

Lookup Tables

The lookup tables used in this study have a chromosome consisting of a table of eight actions indexed by the eight possible ways the opponent's last three moves could have been made. These are called *depth-3* lookup tables. The representation is a string of eight bits, diagrammed in Figure 7.

Opponent's Previous Moves	Response
CCC	C
CCD	D
CDC	D
CDD	C
DCC	D
DCD	C
DDC	C
DDD	D

Fig. 7. A lookup table of the sort used in this study. The opponent's last three actions serve as an index. The representation for lookup tables is the string whose characters are the column of responses, in this case CDDCDCCD.

The evolutionary algorithm uses two-point crossover on the string of eight responses and a mutation operator that picks one action in a random position and changes it.

Artificial Neural Nets

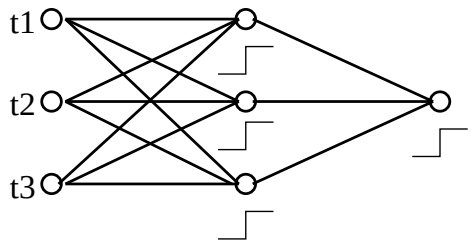


Fig. 8. A feed forward neural net with four 0-1 threshold neurons. The opponent's last three actions are used as inputs. A hidden layer with three neurons is used to drive a single output neuron.

The type of artificial neural nets (ANNs) used in this study are described in [18]. They have a hidden layer of 3 neurons and a single output neuron. All neurons are 0-1 threshold neurons. The topology of the nets is shown in Figure 8. The chromosome for this representation consists of the 12 connection weights: nine for the connections of the three input neurons to the hidden layer and three for the connections of the hidden layer to the output neuron. Neural connection weights were initialized in the range $-1 \leq c \leq 1$.

To create a chromosome, the weights are placed into a linear array. The first nine entries are the input-to-hidden layer connections in groups of three associated with a given hidden neuron. The three remaining values are at the end of the chromosome and represent the connection of the hidden layer to the output neuron. The binary variation operator for this representation is two-point crossover. The point mutation operator adds a number selected uniformly at random in the range $-0.1 \leq x \leq 0.1$ to a connection weight also selected uniformly at random.

IV. Results and Conclusions

The comparison of the evolved neural nets, lookup tables, and finite state machines are facilitated by the following theorem. This theorem is proved constructively in [23].

Theorem 7: Lookup tables and feed-forward neural nets that depend on the same number n of moves by the opponent encode the same strategy space which is itself a subset of the strategy space of finite state machines with 2^n states.

A corollary of this theorem is that the strategies realized by depth-3 lookup tables and the feed forward neural nets in this study are all finite state machines with a single communicating class and no transient states. This has a nice implication: fingerprint identification of these strategies, unlike some finite state strategies such as ripoff and tit-for-tat, is clean without obfuscation by transient states.

In order to compare the evolved populations, the 256 possible depth-3 lookup tables were enumerated. These represent the entire search space for the lookup tables and the neural nets. These lookup tables and all the evolved agents that were not finite state machines were converted into 8-state finite state machines using the construction given in [23]. The 25-point fingerprint, using tit-for-tat as the probe, was computed for each of these machines using the sandpile algorithm. Evolved strategies were classified according to the match between their fingerprint and the depth-3 lookup table fingerprints. For lookup tables and neural nets there was always an exact match. Finite state machines that did not have one of the 256 fingerprints associated with a depth-3 lookup table were classified according to which depth-3 lookup table fingerprint was closest to them using Euclidean distance. Table III summarizes these, showing those lookup tables which appeared at least 50 times for at least one of the representations.

If we call the strategies in Table III the *common* strategies for this experiment, then all common strategies for neural nets were able to answer cooperation with defection at some point. All common strategies coded as lookup tables never answered cooperation with defection – their defection is always in response to the opponent's defection. This type of strategy is called a *nice* strategy[13]. This is not to say the nice/non-nice division between lookup tables and neural nets is absolute, but only strategies that were not common in any of the three representations crossed this divide. This result places a much sharper point on the result found in [9] which found that lookup tables were more cooperative than artificial neural nets. The reason for this remains obscure. On a high level the reason must be rooted in the differences between the details of the two representations because they encode identical strategy spaces. Work in progress suggests that the key fact is that the

Index	LKT	ANN	FSM	Name
0	0	21	78	AllC
13	508	0	8	
15	1702	0	3284	tit-for-tat
25	61	0	35	
27	55	0	1	
28	117	0	3	
43	329	4	2	
47	470	0	13	
59	101	0	0	
61	279	0	0	
69	128	0	10	
73	63	0	0	
77	1834	0	0	
89	140	0	0	
92	84	0	0	
93	1623	0	1	
107	424	0	0	
117	96	0	1	
121	183	0	0	
123	538	0	3	
128	0	1071	0	
136	0	887	0	
151	0	215	0	
159	0	162	0	
185	0	351	0	
200	0	369	0	
232	0	555	0	
247	0	3486	0	
249	0	253	0	
255	0	2069	441	AllD
Remaining FSM:			5575	

TABLE III

A TABULATION, RELATIVE TO THE STRATEGY SPACE OF DEPTH-3 LOOKUP TABLES, OF THE STRATEGIES THAT WERE COMMON FOR AT LEAST ONE OF THE REPRESENTATIONS. ONLY STRATEGIES WITH AT LEAST 50 REPRESENTATIVES FOR ONE OF THE REPRESENTATIONS APPEAR IN THE TABLE. INDEX NUMBERS ARE ARBITRARY, SAVE THAT THOSE BELOW 128 DO NOT DEFECT ON THE ROUND AFTER THE OPPONENT COOPERATES AND THOSE WITH INDEX AT LEAST 128 CAN DO THIS.

probability of generating a tit-for-tat player randomly is much higher for lookup tables than neural nets, at least as they are represented here. A tit-for-tat player responds only to the last action of its opponent and it is much easier, in the current encoding, for lookup tables to ignore all but the last action than for neural nets to do so. Noticing this division into nice and non-nice strategies was a side effect of fingerprinting the strategies, though if it were the only goal it could have been done in a simpler manner.

Recall that the construction of finite state machines from lookup tables given in [23] yields the following fact: strategies coded as lookup tables consist, when transformed into finite state machines, of a single communicating class with no transient states. As noted before, this means that fingerprint identification of lookup table encoded strategies is unambiguous, but this fact has evolutionary implications as well. In [29], [10] it was found that finite state machines use transient states as a means of identifying "kin"; a type of tag or marker. The majority, $5575/9600 = 58.3\%$ of the evolved finite state machines were not strategies that could be coded as depth-3 lookup tables. An informal survey found that many of these used transient states. This ability to identify kin with transient states makes evolutionary options available in the finite state encoding that are not there for lookup tables and the neural nets used in this study.

The simple strategies AllC, AllD, and tit-for-tat were all common for some representation. Tit-for-tat was the most common strategy among the evolved finite state machines and second most common among the evolved lookup tables and was absent in the evolved neural nets. The strategy AllD, on the other hand, was well-represented among the evolved finite state strategies and neural nets but was absent in the evolved lookup tables.

Periodic strategies, other than AllC and AllD, were rare in the evolved finite state populations. This is probably because it is easy for an opponent to learn when a periodic strategy will defect or cooperate and exploit it optimally. The lookup table and neural net strategies cannot implement periodic strategies other than AllC and AllD. This argument would seem to indicate that AllC should not be common for any of the representations since it is the most exploitable and most easily learned strategy. Its appearance can be explained by noticing that it is a single mutation away from tit-for-tat in the finite state representation and can coexist indefinitely with tit-for-tat. The lookup table encoding for tit-for-tat is **CDCDCDCD**, four mutations from AllC, providing additional evidence that the presence of AllC in the other two representations is leakage from tit-for-tat. There are no AllC players in the lookup table populations because it is both exploitable *and* mutationally distant from common strategies.

For furthur analysis of the fingerprints of these evolved strategies, see [11].

V. FUTURE DIRECTIONS

Two pairs of strategies that were substantially different but had identical fingerprints were located: tit-for-tat and ripoff; random and PerCDDC. This suggests that a higher resolution form of fingerprint is required. Three strategies for creating improved fingerprints have been worked out. The first is to extend the fingerprint with information of the same type but based on a finite number of plays rather than on asymptotic behavior (finite time horizon fingerprinting). The second is to include the score of the strategy being fingerprinted against a small list of simple strategies (augmented fingerprints). The third is to use asymptotic fingerprints for more than one probe strategy (multi-probe fingerprints).

All of these approaches have strengths and weaknesses. Finite time horizon fingerprints require that a number of sampling steps be chosen – and the "good" number of samples depends on the complexity of the strategies being fingerprinted. It is also slow. The augmented fingerprint is fast, cheap, and distinguishes tit-for-tat and ripoff but still suffers from an inability to distinguish some strategies. Any individual case of confused strategies can be resolved by adding another strategy to the augmenting scores, but this degrades

the automaticity of the process. Multiple probe strategies are computationally costly but, thus far, appear to resolve identity confusion in a more satisfactory manner than the other two approaches. Recall that TF3T, as a probe, separates random and PerCDDC. This example implies two conjectures. The first, that some probe separates random from any finite state strategy, is obvious. The second conjecture does not have an immediate proof: any pair of strategies with a finite state implementation have a probe that separates them unless they have identical sets of communicating classes.

When prisoner's dilemma is used to model real-world situations the introduction of noise into the fitness function is a common feature. The strategy ripoff gains its advantage by living in a noise-free world; in the presence of noise it is unlikely to arise. This, in turn, suggests that when noise is introduced to any representation, the set of strategies that arise have a different distribution. Pavlov, for example, is an error correcting strategy[1] that deals well with noise but which is also exploitable in a noise-free environment. Fingerprinting is an excellent tool for such an assessment. In the past, simpler metrics such as the number of states used in a finite state representation or level of cooperation have been used [25] to document the impact of noise on the evolution of prisoner's dilemma strategies. Fingerprinting provides far more information about which strategies arise with and without noise. The presence of noise in the fitness function during evolution would also make the fingerprint a much cleaner tool for categorizing strategies. With noise present, strategies are more likely to reach their asymptotic behaviors during fitness evaluation, increasing the match between fingerprint and actual agent character.

An obvious application of fingerprints is to test more representations. This paper is intended primarily to introduce and discuss the fingerprinting tool (fingerprinting code, for finite state machines, is available from the first author on request). The three representations chosen in this study are intended for demonstration; the full set of representations in [9], [5] are a natural target. Fingerprinting the two representations that are not subsets of the finite state strategy space (ISAc lists [1] and Markov chains) will require a good deal of additional code.

Another application is to use fingerprinting to document the impact of changes in the details of the evolutionary algorithm within a single representation. The model of evolution, population size, variation operators and other algorithm details are potential targets. There are a large number of different ways to evolve game playing agents – fingerprinting can be used to explore their impact and classify these evolutionary algorithms into families that produce similar sets of agents.

While the theory of fingerprinting was developed in the context of prisoner's dilemma, the authors are already applying it to the graduate school game[1] and rock-paper-scissors. Preliminary analysis demonstrates that evolved populations of agents for the graduate school game, which reverses the inequality $2C \geq S + T$, are more diverse than prisoner's dilemma populations. On reflection this is not surprising; many prisoner's dilemma populations are dominated by a single strategy while the graduate school game must have two strategies to obtain the maximum score. Since reproduction is occurring with full mixing, it follows that the two types must interbreed and still produce two types. A diversity of more than two types is one of the outcomes observed. In a sense, the graduate school game's second inequality *encourages* unstable genetics.

Fingerprinting generalizes immediately to any k-move simultaneous two-player game. The preliminary results from the graduate school game above suggest that an interesting project, made possible by fingerprinting, is to classify games by their impact on evolutionary dynamics.

VI. Acknowledgments

The first author thanks David Fogel for many helpful conversations on the iterated prisoner's dilemma and would also like to thank the University of Guelph and the National Science and Engineering Research Council of Canada for supporting this work. The second author thanks the National Institute For Mathematical Sciences for its support of this research.

References

[1] D. Ashlock. *Evolutionary Computation for Opimization and Modeling*. Springer, New York, 2006.

[2] D. Ashlock. Training function stacks to play iterated prisoner's dilemma. In *Proceedings of the 2006 IEEE Symposium on Computational Intelligence in Games*, pages 111–118, 2006.

[3] D. Ashlock. Cooperation in prisoner's dilemma on graphs. In *Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Games*, pages 48–55, 2007.

[4] D. Ashlock, W. Ashlock, and G. Umphry. An exploration of differential utility in iterated prisoner's dilemma. In *Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 271–278, 2006.

[5] D. Ashlock and E. Y. Kim. The impact of cellular representation on finite state agents for prisoner's dilemma. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, pages 59–66, New York, 2005. ACM Press.

[6] D. Ashlock, E. Y. Kim, and W. K. vonRoeschlaub. Fingerprints: Enabling visualization and automatic analysis of strategies for two player games. In *Proceedings of the 2004 Congress on Evolutionary Computation*, volume 1, pages 381–387, Piscataway NJ, 2004. IEEE Press.

[7] D. Ashlock and E.Y. Kim. Techniques for analysis of evolved prisoner's dilemma strategies with fingerprints. In *Proceedings of the 2005 Congress on Evolutionary Computation*, volume 3, pages 2613–2620, Piscataway, NJ, 2005. IEEE Press.

[8] D. Ashlock, E.Y. Kim, and L. Guo. Multi-clustering: avoiding the natural shape of underlying metrics. In C. H. Dagli et al., editor, *Smart Engineering System Design: Neural Networks, Evolutionary Programming, and Artificial Life*, volume 15, pages 453–461. ASME Press, 2005.

[9] D. Ashlock, E.Y. Kim, and N. Leahy. Understanding representational sensitivity in the iterated prisoner's dilemma with fingerprints. *IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, 36(4):464–475, 2006.

[10] 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.

[11] W. Ashlock. Why some representations are more cooperative than others for prisoner's dilemma. In *2007 IEEE Symposium on the Foundations of Computational Intelligence*, pages 314–321, Piscataway, NJ, 2007. IEEE Press.

[12] W. Ashlock and D. Ashlock. Changes in prisoner's dilemma strategies over evolutionary time with different population sizes. In *Proceedings of the 2006 Congress On Evolutionary Computation*, pages 1001–1008, Piscataway, NJ, 2006. IEEE press.

[13] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.
[14] R. Axelrod and W. D. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, 1981.
[15] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming : An Introduction*. Morgan Kaufmann, San Francisco, 1998.
[16] S. Y. Chong and X. Yao. Behavioral diversity, choices and noise in the iterated prisoner's dilemma. *IEEE Transaction on Evolutionary Computation*, 9:540–551, 2005.
[17] N. Franken and A. P. Engelbrecht. Particle swarm optimization approaches to coevolve strategies for the iterated prisoner's dilemma. *IEEE Transaction on Evolutionary Computation*, 9:562–579, 2005.
[18] S. Haykin. *Neural Nets, a Comprehensive Foundation*. Macmillan College Publishing, New York, 1994.
[19] M. Hemesath. Cooperate or defect? Russian and American students in a prisoner's dilemma. *Comparative Economics Studies*, 176:83–93, 1994.
[20] 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.
[21] T. W. Hungerford. *Albegra*. Springer-Verlag, New York, 1974.
[22] H. Ishibuchi and N. Namikawa. Evolution of iterated prisoner's dilemma game strategies in structured demes under random pairing in game playing. *IEEE Transaction on Evolutionary Computation*, 9:540–551, 2005.
[23] E. Y. Kim. *Analysis of Game Playing Agents with Fingerprints*. PhD thesis, Iowa State University, 2005.
[24] J. F. Miller and S. L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transaction on Evolutionary Computation*, 10(2):167–174, 2006.
[25] 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.
[26] S. I. Resnick. *Adventures in Stochastic Processes*. Birkhauser, Boston, 1992.
[27] D. Roy. Learning and the theory of games. *Journal of Theoretical Biology*, 204:409–414, 2000.
[28] K. Sigmund and M. A. Nowak. Evolutionary game theory. *Current Biology*, 9(14):R503–505, 1999.
[29] E. A. Stanley, D. Ashlock, and L. Tesfatsion. Iterated prisoner's dilemma with choice and refusal. In Christopher Langton, editor, *Artificial Life III*, volume 17 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 131–176, Reading, 1994. Addison-Wesley.
[30] W. K. vonRoeschlaub. Automated analysis of evolved strategies in iterated prisoner's dilemma. Master's thesis, Iowa State University, 1994.