Niche Strategies

The Prisoner's Dilemma Computer Tournaments Revisited

Robert E. Marks

Graduate School of Business,
Stanford University,
California 94305-5015
(until mid-August 1993)

and

Australian Graduate School of Management,
University of New South Wales,
P.O. Box 1,
Kensington NSW 2033, Australia

Internet: `r.marks@unsw.edu.au`

An earlier version of this paper was
presented at the 1988 Australasian meetings
of the Econometric Society,
ANU, Canberra, August 1988,
under the title, "Breeding hybrid strategies:
the Prisoner's Dilemma computer tournaments revisited."

KEYWORDS: Axelrod, genetic-simulation, machine-learning, repeated-Prisoner's-Dilemma, strategy-modelling

ABSTRACT: The repeated Prisoner's Dilemma has proved to be a rich area of study for examining the tension between competition and coöperation. Computer tournaments—such as Axelrod's and Fader and Hauser's—have sought to obtain "effective" strategies in repeated games from diverse entrants, as a way of generating robust results from a broad range of competing strategies. The development of Holland's Genetic Algorithm has enabled researchers to dispense with open tournaments to obtain such diversity, and to derive strategies almost certainly closer to globally optimal, given the environment of rivals.

This paper generalises a process reported by Axelrod in describing how the genetic algorithm has been used to generate winning strategies of specified complexity in a repeated 2-person Prisoner's Dilemma, against two specific "niches" of competing strategies, from Axelrod's second computer tournament. It also derives winning strategies in "noisy" PD, in which competitors are not certain of their opponents' moves. These strategies exhibit some of the characteristics seen in Rapoport's famous Tit for Tat.

In an analysis of strategic complexity and its relationship to strategic success, the paper examines finite automata in an attempt to simplify the strategies obtained, which are conditioned on 1-, 2-, and 3-round memory of the moves and outcomes of previous rounds. Winning strategies are represented as finite automata.

ABBREVIATIONS:

GA	Genetic algorithm
PD	Prisoner's Dilemma.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Repeated-Play Strategies

MICRO-ECONOMICS has recently been enriched by studies of strategic behaviour among small numbers of competitors, such as in oligopolistic markets. The small firm in a purely competitive market faces a horizontal demand curve which it is powerless to shift up; the monopolist faces a downwards-sloping demand curve which it too is powerless to shift (although it can decide its price, somewhere along the curve). But most firms are faced with competitive environments that do change and respond to their actions, as their rivals adjust their responses. Studies of this strategic phenomenon have been in terms of the dynamic adjustment of the competitors' behaviour, and have been facilitated by the insights from game theory (Ulph 1987).

The strategic behaviour between two competitors has been extensively studied in simple two-person games, the most insightful of which has been the Prisoner's Dilemma (PD). In a one-shot game the PD demonstrates how the logic of competition, in the absence of trust or enforceable pre-commitment, results in a Nash solution of non-coöperation that is Pareto-inferior to the coöperative solution.

		Player 2	
		C	*D*
Player 1	*C*	4, 4	0, 5
	D	5, 0	2, 2

TABLE 1. Payoff matrix for the Prisoner's Dilemma

In Table 1, *x*,*y* means that Player 1's payoff is *x* units and Player 2's is *y* units. *C* corresponds to coöperating, *D* to defecting, The Reward *R* for mutual coöperation is 4; the Punishment *P* for mutual defection is 2; the Temptation *T* of cheating is 5; and the Sucker's payoff *S* of being cheated is 0 (Axelrod 1984).[1]

1. As discussed below, these values have been chosen with repeated play in mind: repeated cooperation pays more than alternating cooperation/defection.

In a once-off PD game, the dominant strategy is to defect, despite a higher payoff for coöperation, because of the reward of cheating and the penalty of being cheated. For each player, defecting (D) dominates coöperating (C), and the outcome is mutual defection (D,D) instead of mutual coöperation (C,C), which would result in each player receiving the Punishment for mutual defection, instead of the greater Reward for mutual coöperation. This outcome occurs despite both players' clear knowledge of the gains from coöperation.

In a repeated PD game, however, the higher payoff to coöperation may result in strategies different from the "always defect" of the single game (the Folk Theorem). By breaking the logical imperative of mutual defection inherent in the static, one-shot PD, the repeated PD—in which the players repeatedly face each other in the same situation—can model the possibility of *learning* on the part of the players, which may result in mutual coöperation or some more complex strategy on their part, as they learn more about the type of responses (such as punishment for defection) they can expect from each other.

The process of learning (for which the repeated PD provides a framework for studying) is one that economics has not had great success in modelling (Simon 1978). Despite the efforts of some economists (Nelson and Winter 1982) to study the *processes* of economics activity, mainstream economics has preferred to assume away the adaptive process of learning than to let ad hoc theories of learning intrude on the axiomatic consistency of the neoclassical framework.[2]

An early analysis of successful strategies in the repeated PD (Luce and Raiffa 1957, pp. 97–102) suggested that continued, mutual coöperation (C,C) might be a viable strategy, despite the rewards from defection, but for twenty years no stronger analytical results were obtained for the repeated PD.

In the late 1970s, political scientist Robert Axelrod, in an investigation of the emergence of coöperative behaviour and social norms in Hobbesian societies, hit upon the idea of exhaustively pitting strategies for the repeated PD by coding them into computer algorithms. In Axelrod (1984) he reported the results of two tournaments between strategies played on computer. In essence his call for strategies (coded as algorithms) was an attempt to search the strategy space by asking

2. The theory of "rational expectations," for instance, assumes that any learning necessary for optimisation has already occurred—or that it occurs instantaneously and costlessly, as necessary. Such empirical tests as Hofstadter's Luring Lottery (1985, p. 751) demonstrate that, as a description of the behaviour of even an intelligent and well informed group of people—readers of his *Scientific American* column—the theory is inadequate.

researchers in diverse disciplines to devise strategies.

Since the PD is a non-zero-sum game, to win the repeated game is not a function of always snatching, as it were, the larger share of the cake—since the struggle to seize the larger slice will reduce the size of the cake overall, to the ultimate loss of both players. So, although the strategy of Always Defect will never be beaten—will always take the larger share in each game—it will not score the highest aggregate across many games—the largest total quantity of cake, as it were. In a positive-sum game, such as PD, there can be no optimum optimorum: instead, the winner depends upon the population (of other strategies) against which it is pitted. What Axelrod was seeking was a robust "winning strategy" in the repeated game; such a strategy had evaded analytical efforts. The scores for the four possible outcomes in Table 1 were chosen so that $(T+S) < 2 \times R$ (where $T = \text{Temptation}$, $S = \text{Sucker's}$, and $R = \text{Reward}$), so that the alternating pattern of (C,D), (D,C), (C,D), (D,C), . . . results in a lower average score than the pattern of mutual coöperation of (C,C), (C,C), (C,C), . . .

As is now widely known, Axelrod's tournaments revealed that one very simple strategy is robust in the repeated PD: Rapoport's Tit for Tat, which coöperates on the first round of the repeated game, and then mimics its opponent. Although it never does better than tying in a one-on-one tournament against any opponent, when pitted against a "nasty" strategy, such as "Always Defect," it does almost as well, itself defecting on every round but the first, but at the cost of the aggregate score. With regard to its score rather than whether it wins or not, Tit for Tat is robust. When played against itself, both players' aggregate score is a maximum, since every round will then be (C,C), a result which resembles collusion, although each player's decisions are made independently of the other's. One motivation for this study is to explore whether, with no limit on strategic complexity, Tit for Tat can be soundly bettered.

Axelrod's tournaments and later tournaments modelling a three-person price war (Fader and Hauser 1988) were an attempt to pit as wide a variety of strategies against each other as possible, in order to derive more robust results and insights than would follow with a small set of strategies. Mathematically, the problem of generating winning strategies is equivalent to solving a multi-dimensional, non-linear optimisation with many local optima. In biological-evolution terms, it is equivalent to selecting for fitness. Indeed, in a footnote, Cohen and Axelrod (1984, p. 40) suggest that

> One possible solution may lie in employing an analogue of the adaptive process used in a pool of genes to become increasingly more fit in a complex environment. A promising effort to convert the main characteristics of this process to an heuristic algorithm is given by John

Holland (1975 [1992]). This algorithm has had some striking preliminary success in the heuristic exploration of arbitrary high dimensionality nonlinear functions.

Following up his own suggestion, Axelrod has used Holland's Genetic Algorithm (GA) (see Section 3 below) to "breed" strategies in the two-person repeated PD game (Axelrod 1987). He reports that the GA evolved strategy populations whose median member was just as successful as Tit for Tat, whom they closely resembled. (In 95% of the time, the evolved rules make the same choice as would Tit for Tat in the same situation.) In some cases the GA was able to evolve highly specialised adaptations to a specific population of strategies which perform substantially better than does Tit for Tat in that situation. This paper attempts to replicate Axelrod's work, and to examine how the GA could be used in the breeding of strategies to such problems as the two-person PD with uncertainty ("noise") (Nalebuff 1987) and the three-person PDs of the price war (Fader and Hauser 1988).

The advent of GAs (and machine learning) means that a much more exhaustive set of potentially winning strategies can be generated by a single researcher, without the combined efforts of many competitors. This is because, within any given degree of "strategic complexity" (see Section 2.1 below), *any* potential strategy is grist to the GA's mill, and will eventually be tested if it is a contender for best strategy, given the environment of competitors.

2. Modelling Strategies for Repeated Games

2.1 Finite Automata and Moore Machines[3]

THE use of finite automata in repeated games derives from Aumann (1985), and has since been used by several authors (Neyman 1985; Radner 1986; Rubinstein 1986; Ben-Porath 1987; and Miller 1989). Originally, it was formulated in an attempt to make Simon's (1972) concept of "bounded rationality" operational, by marrying the theory of finite automata with game theory, to derive a measure of strategic complexity. An additional reason was to see whether putting a bound on the complexity of players' strategies could induce coöperative behaviour in the finitely repeated Prisoner's Dilemma. Neyman shows that this is so.

3. See Marks (1992*a*) for a survey of this topic.

Rubinstein, in an infinitely repeated world, describes a world in which players select Moore machines (Moore 1956) instead of explicit strategies. A Moore machine is a device in which the player's next move is contingent on the existing state of the machine, which in turn is a function of the previous state of the machine (at the previous round) and the other player's previous move, through a transition (or next-state) function. (The initial state and the set of all feasible internal states of each machine must be stated at the outset, along with the set of all feasible moves and the transition function and the "action"—or output—function.) If both players in a two-person game have chosen Moore machines, then the game can continue between the machines, which will generate states and moves as the repeated game progresses. Although previous researchers have used finite automata theory to develop theoretical results about strategies in repeated games with limits on strategic complexity, Moore machines also provide a way of using techniques of machine learning to search for robust strategies in repeated games, thus obviating the need for further computer tournaments among submitted strategies.

We can formalise a Moore machine, and provide some examples. Let Q_i be a finite set, the set of possible internal states of player i's automaton, and let S_i and S_j denote the finite sets of actions or moves for players i and j, respectively. If in round t the state of player i's machine is $q_i(t) \in Q_i$ and player j's move is $s_j(t) \in S_j$, then at round $t+1$ the state of player i's machine, $q_i(t+1)$, will be

$$q_i(t+1) = g_i[q_i(t),\ s_j(t)],$$

and player i's move (or action) in round $t+1$, $s_i(t+1) \in S_i$, will be

$$s_i(t+1) = f_i[q_i(t+1)].$$

The quadruple $\langle Q_i,\ \bar{q}_i,\ f_i,\ g_i \rangle$ constitutes player i's automaton, where $\bar{q}_i \in Q_i$ is the initial state of the machine, where f_i is the action function, $f_i : Q_i \rightarrow S_i$, and where g_i is a the next-state (or transition) function, $g_i : Q_i \times S_j \rightarrow Q_i$. The number of elements in Q_i is the size of the automaton.

Rubinstein presents a series of "transition diagrams" for machines which play familiar strategies in the repeated Prisoner's Dilemma. The nodes of these diagrams correspond to what Rubinstein calls "states," one of which is the "Start," but his "states" are not the same as ours below: as we shall see, they are sets of our states. The letters C or D immediately *beneath* each node show the machine's move associated with that node. The letters C and/or D immediately *above* each arc correspond to the others player's move, after which the machine moves to the new node at the arrowed end of the arc.

For instance, a machine which plays *C* constantly (Always Coöperate) can be described as

$$Q = \{ q^* \},\ \overline{q} = q^*,\ f(q^*) = C, \text{ and } g(q^*, \cdot) \equiv q^*.$$

Rubinstein depicts it as Figure 1.

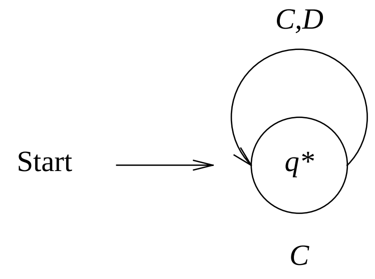

Figure 1. The “Always Coöperate” Moore Machine

Rapoport’s strategy, Tit for Tat, can be described as

$$Q = \{ q^C, q^D \},\ \overline{q} = q^C,\ f(q^s) = s \text{ and } g(q,s) = q^s \text{ for } s = C, D.$$

Its transition diagram is given by Figure 2.

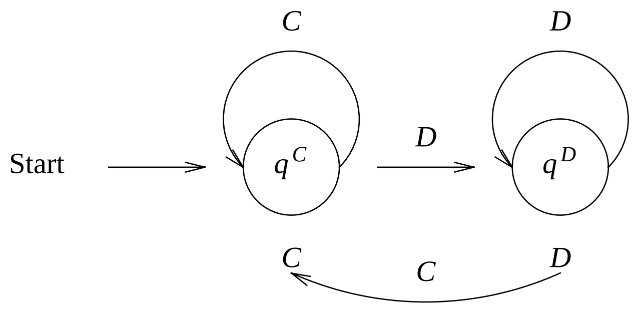

Figure 2. The “Tit for Tat” Moore Machine

The strategy of playing *C* until the other player plays *D* and then punishing him for three periods before returning to coöperation requires at least a four-node machine, as depicted in Figure 3. It can be described as

$$Q = \{ q, p_1, p_2, p_3 \},\ \overline{q} = q,\ f(q) = C,\ f(p_h) = D,\ (h = 1,2,3),$$
$$g(q,C) = q,\ g(q,D) = p_1,\ g(p_h, \cdot) \equiv p_{h+1}, \text{ and } g(p_3, \cdot) \equiv q.$$

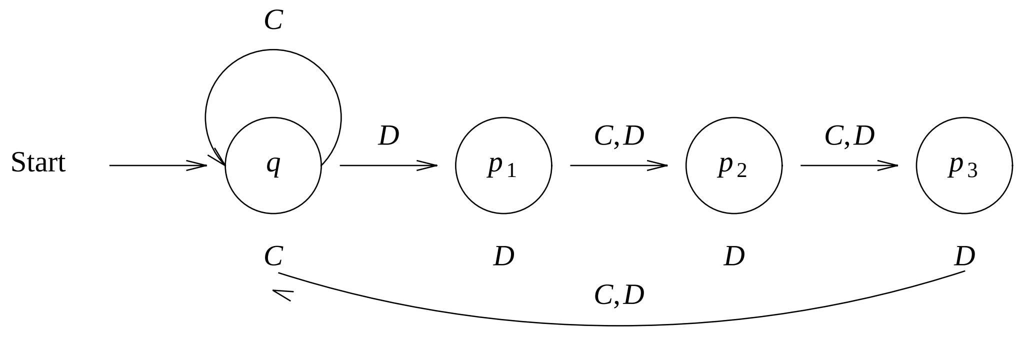

Figure 3. A Four-Node Moore Machine

It is possible to represent these machines by strings, together with rules describing the transition and action functions. Each position (or locus) on the string corresponds uniquely to a state. The action function is simply a mapping from the position (or locus) on the string to the character or allele (*C* or *D*) at that position. The transition function will result in a new position (or state), contingent on the previous position and the other player's previous move.

For instance, in the Always Coöperate machine of Figure 1, there is only one node, which always results in *C*. Thus, the string representation of this machine might be the string *C*, and, whatever the previous move of the other player, the machine's response would be an unchanging *C*. For Tit for Tat there must be at least two elements in the string, one corresponding to the other player's coöperating in the previous round, and the other corresponding to his defecting in the previous round. The first results in the machine's response of *C*, the second in *D*. Thus, the string representation of Tit for Tat might be, say, *CD*, where *C* corresponds to node 1 and *D* corresponds to node 2, as in Figure 2. The algorithm would tell us to look at node 1 for our next move if the other player's previous move was *C*, and to look at node 2 for our next move if the other player's previous move was *D*. The four-node strategy of Figure 3 might be represented by the string *CDDD*; this strategy is not as simple as the previous two; the transition function, for instance, is not simple, although the Rubinstein transition diagram can be followed without too much difficulty. It appears that a strategy which has no memory (Figure 1) requires one node, that a 1-round memory (Figure 2) requires two nodes, and that a 3-round memory (Figure 3) requires four nodes.

The size of an automaton is the number of states it has. The complexity of a strategy is defined by Ben-Porath (1987) as the minimal size of the automaton that can implement it. From Rubinstein's transition diagrams above, it appears that Always Coöperate is of lowest strategic complexity, followed by Tit for Tat, and that Figure 3 depicts a strategy of higher complexity. But we need to define the concept of *a state* more rigorously than does Rubinstein if the measure is to have meaning, as we see below. Moreover, as Radner (1986) notes, this measure does not take account of the complexity of the action function and the transition function—what Gottinger (1983, p. 127) calls the tradeoff between structural complexity and computational complexity.

Nonetheless, Ben-Porath's measure of strategic complexity raises the question:

> *Given any level of strategic complexity, what is the most successful strategy in competing against a given environment of strategies?*

Tit for Tat has proved itself to be, at a low level of strategic complexity, extremely robust against a wide range of opponents. This raises another question:

> *With no limit on strategic complexity, can Tit for Tat be soundly bettered?*

2.2 Strategies Contingent upon Previous Actions

As suggested in the previous section, and following Axelrod (1987), we model the actions of the players in the repeated game as contingent upon the actions of both players in previous rounds. That is, the next-state function g_i has as its argument the state q_i, which is simply defined as the combination of players' actions in the previous rounds. We consider three levels of strategic complexity. The lowest corresponds to looking back at the most recent round only: *1-round memory*; the highest corresponds to looking back at the three most recent rounds: *3-round memory*; we also consider *2-round memory*. For each round there are four states—(C,C), (C,D), (D,C), and (D,D), where by convention (X,Y) describes a round in which the player's own move was X and the other player's move was Y—so there are 4^r possible states for r-round memory. For 1-round memory, there are 4 possible states; for 2-round memory 16; and for 3-round memory, 64. The number of states is an upper limit of strategic complexity, as Figure 2 demonstrates. For exposition's sake, we consider 1-round-memory strategies in detail.

A strategy is characterised by its action function, a mapping from the possible states of recent history to the two possible actions in the next round, C or D. For a 1-round memory, this mapping can be thought of as a binary string of length 4 bits, where 0 models C and 1 models D. Each of the 4 positions on the string corresponds to one state q of recent history, and the rule is: coöperate (C) or defect (D) on the next round depending solely on whether there is a zero or one, respectively, at the position (or locus) q of the string: the allele at locus q maps state into action.

In general, we calculate the position or state q from the formula:

$$q = \sum_{i=1}^{r} 4^{(r-i)}\ [own(i) + 2 \times other(i)], \tag{1}$$

where r is the number of rounds remembered, $own(i)$ is the action taken by this player i rounds ago, and $other(i)$ is the action taken by the other player i rounds ago. (Recall that C is 0 and D is 1.) So mutual coöperation for the last three rounds (r = 3) corresponds to q = 0, and mutual defection for the last three rounds to q = 1+2+4+8+16+32 = 63. For 1-round memory, this equation collapses to:

$$q = own(1) + 2 \times other(1), \tag{2}$$

which means that the four states are as shown in Table 2.

C, C	$\rightarrow$	$q = 0$
D, C	$\rightarrow$	$q = 1$
C, D	$\rightarrow$	$q = 2$
D, D	$\rightarrow$	$q = 3$

TABLE 2. The Mapping from History to State, for One-Round Memory

For 1-round memory, there are 2^4 = 16 possible mapping strings or strategies in this two-action repeated game. As seen in Table 3, these can readily be examined exhaustively, without need for the GA described below, but 3-round memory leads to 2^{64} possible mapping strings, an altogether different search task. In general, for this two-action game, the number of possible mapping strings is 2^{4^r}.

From Table 3, we can readily recognise some strategies: Strategy 0 is Always Coöperate, since whatever the last round moves of the players the next move is 0, or C; Strategy 15 is Always Defect; Strategy 3 is Tit for Tat, since C follows the other player's C last round (q = 0 and 1), and D follows the other player's D last round (q = 2 and 3).

Strategy 5 results in unchanging C or D, depending on the first move, which raises the question of how to act in the first round (or rounds), when there is no history of previous moves. Again following Axelrod (1987), we add additional bits to the mapping string to represent *phantom memory*, and search among the augmented strings for high-scoring strings that include the initial moves. With 1-round-memory strategies, there is only one phantom round, which requires 2 bits for its representation, since there are 4 possibilities for the phantom state. Higher-

Number	String
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

TABLE 3. All Possible One-Round-Memory Strategies

round memory would require more phantom memory states: 3-round-memory strategies require 6 bits of phantom memory, 2 per round.

A 3-round-memory string chosen at random might resemble that in Table 4.

10010100 10100101 00110101 00101110 10100010 10010101 11010100 10101011

TABLE 4. A Random Three-Round-Memory String

From equation (1) with r = 3, a history of both players coöperating for each of the last three rounds ($\{CC, CC, CC\}$) corresponds to $q = 0$. The string in Table 4 will result in a defection in the next round: the position or state q_0 for this string maps into 1 (= D), since $f\,(q_0) = 1$, where the action function $f\,(q_s)$ is the value of the state $q_s \equiv s$, which is simply given by the sth position (or locus) on the string, counting rightwards from zero rather than one. Similarly, since $f(q_{63}) = 1$, a history of mutual defection for the past three rounds will result in a further defection on the part of this individual.

2.3 Strings as Particular Strategies

It is difficult to recognise any but the simplest rules from examination of individual strings. Clearly, a string of ones would result in “always defect” behaviour, and a string of zeroes would result in “always coöperate” behaviour, but such behaviour may also result from other strings. This follows because each string maps from a history that includes its own behaviour, and if it never coöperates (in the first example) or never defects (in the second) then some feasible states will never occur, and the corresponding positions on the string will never be used—those corresponding to its own coöperation in the first example and its own defection in the second.

If a string of ones is *sufficient* for Always Defect, what is a *necessary* string for this behaviour, assuming any behaviour on the part of the other player over the past rounds? From Table 2, the only relevant states with 1-round memory are q = 1 and 3, since the other states require own behaviour of C. Consequently, the necessary string for Always Defect with 1-round memory is #1#1, where the # symbol is used to indicate that the bit at that position is irrelevant to the desired behaviour. Consider equation (1) with $r = 3$, for 3-round memory. If own behaviour is Always Defect, then $own(i) = 1,\ i = 1,2,3$. Hence, $q \geq 1{+}4{+}16 = 21$, counting from zero. This means that the first 21 alleles are irrelevant. As we shall see when examining the mapping in Table 8, all but eight states are in fact irrelevant (those associated with the 2^3 possible actions of the other player over the past three rounds). The necessary strings for Always Defect are shown in Table 5.

```
                              #1#1
                         #####1#1 #####1#1
######## ######## #####1#1 #####1#1 ######## ######## #####1#1 #####1#1
```

TABLE 5. Necessary Strings for “Always Defect”

Similarly and symmetrically, the necessary strings for Always Coöperate are shown in Table 6, where the zeroes correspond to coöperation in the eight possible states.

0#0#

0#0##### 0#0#####

0#0##### 0#0##### ######## ######## 0#0##### 0#0##### ######## ########

TABLE 6. Necessary Strings for “Always Coöperate”

These two simple cases suggest that the strategic complexity of Always Defect and Always Coöperate is 8 in the case of 3-round memory. When we compare the representation of Table 6 with the Moore machine of Figure 1, which has only one node, we can see that Rubinstein’s node includes eight of our states. The action function of Always Coöperate is $f(\cdot) = C$, whatever the previous move of the other player, and with the 2^3 possible moves of the opponent, there must be eight possible states. For a rigorous approach to the concept of complexity, see Futia (1977) and Gottinger (1983, pp. 3–75; 1987). Suffice it here to note that we shall consider strategies with up to 64 independent states in Section 4, although the internal logic of a strategy’s own possible moves will usually reduce this number considerably, as we have seen in Tables 5 and 6.

The necessary strings for Tit for Tat are shown in Table 7, for 1-, 2-, and 3-round memories. In the latter two strings, positions marked with a # correspond to a history in which own behaviour in the previous two or three rounds did not obey Tit for Tat and so are irrelevant. For 1-round memory this possibility does not arise.

0011

00####00 11####11

00####00 ######## ######## 00####00 11####11 ######## ######## 11####11

TABLE 7. Necessary Strings for “Tit for Tat”

The complete mapping from the last three rounds to the 64 states or loci is shown in Table 8, which is the 3-round memory equivalent of Table 2 for 1-round memory. For instance, $q = 6$ (counting from zero) is equivalent to the state of history given by: (C,D) followed by (D,C) followed by (C,C) (or $\{CD,DC,CC\}$), in which the player’s own behaviour in the second last and last rounds was to mimic the other player’s behaviour of the third last and second last rounds, respectively. For this mimicry to continue, the next move of the own player must mimic the other

q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
i=3	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD
i=2	CC	CC	CC	CC	DC	DC	DC	DC	CD	CD	CD	CD	DD	DD	DD	DD
i=1	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC

q	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
i=3	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD
i=2	CC	CC	CC	CC	DC	DC	DC	DC	CD	CD	CD	CD	DD	DD	DD	DD
i=1	DC	DC	DC	DC	DC	DC	DC	DC	DC	DC	DC	DC	DC	DC	DC	DC

q	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
i=3	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD
i=2	CC	CC	CC	CC	DC	DC	DC	DC	CD	CD	CD	CD	DD	DD	DD	DD
i=1	CD	CD	CD	CD	CD	CD	CD	CD	CD	CD	CD	CD	CD	CD	CD	CD

q	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
i=3	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD	CC	DC	CD	DD
i=2	CC	CC	CC	CC	DC	DC	DC	DC	CD	CD	CD	CD	DD	DD	DD	DD
i=1	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD	DD

TABLE 8. The Mapping from History to State, for Three-Round Memory

player's move in the most recent round, which was *C*; so $f\,(q_6)$ must equal 0, as the string of Table 7 shows. This behaviour is identical with Tit for Tat.

From Figure 2, we can exhaustively determine all possible 3-round histories associated with each of Rubinstein's nodes, in order to see the correspondences between his nodes and our states. We see that node q_C = states $\{0,1,6,7,24,25,30,31\}$, and that node q_D = states $\{32,33,38,39,56,57,62,63\}$. Since Tit for Tat is symmetric in terms of *C* and *D* (except for its initial state), we should not be surprised that each node corresponds to an equal number of states; we shall see below that an asymmetric strategy results in uneven numbers of correspondences.

We have modelled the strategies as mappings from the history of the last three rounds to the own behaviour in the next round. But what is to be done during the first three rounds of the game, when there is not yet a complete 3-round history to map from? We could mandate a pattern of behaviour, such as Always Coöperate, for each strategy during the first three rounds, but following Axelrod (1987) we let six additional bits on the string provide the phantom memory for each strategy. (We need six additional bits so that each of the six phantom decisions—three per player—is coded for.) The result of this is that each 3-round-memory strategy is coded as a binary string of length 70, but we shall speak of the first 64 bits as the essence of the strategy, as in the figures above.

Each string can then be evaluated by its success in playing a repeated game, either against an unchanging environment or *niche*, or against a changing environment of similar rivals. The next section describes how the GA can be used to derive ever more successful strings in an example of machine learning. The score or "fitness" of any individual is calculated from a series of games of 151 rounds' length following Axelrod (1984, 1987) against any environment of strategies, using the pay-off matrix of Table 1. Although this is a finite repeated game, in which end-game strategies might result in the one-shot Nash "Always Defect," the memory limit precludes such behaviour.

As Axelrod (1987) points out, each string serves a dual purpose: it is the complete description of the individual, and it provides the information which can be transformed into the next generation if it is "fit" enough, that is, if it performs sufficiently well in the repeated game. He also notes that to search all possible individual 3-round-memory strings exhaustively, at a rate of 100 per second, is "out of the question"—fewer than 1% of this number could have been checked since the beginning of the universe.

3. Genetic Algorithms

IN THE previous section we have seen how strategies (sets of rules) for playing repeated games of the Prisoner's Dilemma can be represented as bit strings of zeroes and ones, each locus or substring (or gene) along the string mapping uniquely from a contingent state—defined above by all players' moves in the previous round or rounds of the repeated game—to a move in the next round, or a means of determining this next move. This representation allows us to use a recently developed optimisation technique—the Genetic Algorithm[4] (GA)—to develop what is in effect a machine-learning process to search for strings which are ever more successful at playing the repeated game. As explained by De Jong (1990), our process can be classified as an example of the Pitt approach to machine learning, in which each string is evaluated for evolutionary fitness (its score in the repeated game), and this score is used to generate a new set of strings. The particular GA we use is Grefenstette's GENESIS (1987).

We describe these strings as "chromosomes" because GAs use selection and recombinant operators—crossover and mutation—derived by analogy from population genetics in order to generate new sets of strings (a new generation of "offspring") from the previous set of strings. Brady (1985) notes that "during the course of evolution, slowly evolving genes would have been overtaken by genes with better evolutionary strategies," although there is some dispute about the extent to which such outcomes are optimal (Dupré 1987). The GA can be thought (Bethke 1981) as an optimisation method which overcomes the problem of local fitness optima, to obtain optima which are almost always close to global. Moreover, following biological evolution, it treats many candidate solutions (individual genotypes) in parallel, searching along many paths of similar genotypes at once, with a higher density of paths in regions (of the space of all possible solutions) where fitness is improving: the "best" individual improves in fitness and so does the average fitness of the set of candidates (the population).

Hereditary models in population genetics define individuals solely in terms of their genetic information: the genetic structure of an individual—or genotype—is represented as strings of chromosomes consisting of genes, which interact with each other to determine the ultimately observable characteristics—or phenotype—of the individual. A population of individuals can be viewed as a pool of genetic information. If all individuals in the population have equal probability of mating and producing offspring, and if the selection of mates is random, then the information in the gene pool will not change from generation to generation. But environmental factors affect the fitness of phenotypes of individuals, and hence affect the future influence of the corresponding genotypes in determining the characteristics of the gene pool: the principle of natural selection, which results in a changing gene pool as fitter genotypes are exploited. Natural selection can be viewed as a search for coädapted sets of substrings which, in combination, result in better performance of the corresponding phenotype (the individual's behaviour) in its environment.

4. For an introduction to GAs, see Goldberg (1988).

Schaffer and Grefenstette (1988) argue that the theory of GAs derived by Holland (1992) predicts that substrings associated with high performance will spread through the new populations of bit strings. Paraphrasing Holland (1984), a GA can be looked upon as a sampling procedure that draws samples from a potential set *T*. With each sample is associated a value, the fitness (or score) of the corresponding genotype (or fundamental hereditary factors). Then the population of individuals at any time is a set of samples drawn from *T*. The GA uses the fitness (scores) of the individuals in the population at each generation to "breed" and test a new generation of individuals, which may include the best individuals from the previous generation. The new generation is "bred" from the old using genetic operators: selection of parents according to their fitness, cross-over of genetic material from both parents, and random mutation of bits. This process progressively biases the sampling procedure towards the use of combinations of substrings associated with above-average fitness in earlier generations (that is, sample individuals characterised by higher scores because their behaviours are "better"), so the mean score of successive generations rises owing to selective pressures. A GA is all but immune to some of the difficulties that commonly attend complex problems: local maxima, discontinuities, and high dimensionality.

The GA is an answer to the problem of obtaining a robust and efficient use of information contained in a limited amount of experience: the difficulty is the low level of confidence that can be placed on inferences from limited samples. The GA, so Schaffer and Grefenstette (1988) argue, results in a near-optimal trade-off between exploration (gaining more reliability from more experience) and exploitation (using the information from past experience to generate new trial structures). The GA can be described in essence as (1) producing the initial population $P(0)$ (randomly or using some seeding method); (2) evaluating each trial structure of the population $P(t)$ at any time *t*; (3) selecting the fittest trial structures, as measured by their scores (in our case in the repeated games); (4) applying the genetic recombination operators to the selected parent structures to produce the offspring generation, $P(t+1)$; (5) testing against a stopping rule—if YES, then stop, if NO, then return to stage (2).

The initial population $P(0)$ is usually chosen at random, but can be constructed to contain heuristically chosen initial strings. In either case, the initial population should contain a wide variety of structures. The structures of the population $P(t+1)$ are chosen from the population $P(t)$ by a randomised "selection procedure" that ensures that the expected number of times a structure is chosen to be a "parent" is proportional to that structure's performance, relative to the rest of the population. That is, if unique structure x_j has twice the average performance of all the structures in $P(t)$, then x_j is expected to appear twice in the mating pool. At the end of the selection procedure, the mating pool contains exact duplicates of the selected structures in population $P(t)$, in proportion to their share of the aggregate of fitness scores.

Although realisations of the GA differ in their methods of survival selection, of mate selection, and of determining which structures will disappear, and differ in their size of population and their rates of application of the different genetic operators, all exhibit the characteristic known as *implicit parallelism.* Any structure or string can be looked at as a collection of substring components or schemata which together account for the good or bad performance of the individual structure. Then Holland's Schema Sampling Theorem (Holland 1992, Schaffer and Grefenstette 1988) demonstrates that schemata represented in the population will be sampled in future generations in relation to their observed average fitness, if we can assume that the average fitness of a schema may be estimated by observing some of its members. (Note that the number of schemata being sampled is greater than the number of individual structures of the population being evaluated.) GAs gain their power by searching the space of all schemata and by quickly identifying and exploiting the combinations which are associated with high performance.

This can be formalised. A population of binary structures of length L bits can be viewed as points in an L-dimensional space. GAs search for better structures by focusing on partitions (hyperplanes) of this space associated with good performance or high fitness. A kth order hyperplane ($0 \leq k \leq L$) is defined as an ($L-k$)-dimensional subspace, and is specified by assigning values to only k of the L string positions or loci, the rest being filled with the # symbol. Let H be a hyperplane in the representation space. Let $M(H,t)$ denote the number of individual structures in $P(t)$, the population at time t, that are members of the hyperplane H. For example, y_2 below is a member of the hyperplane #1001###. Holland (1992) has shown that the effect of selection alone (in the absence of other genetic operators) is that

$$M(H,t+1) = \frac{\mu(H,t)}{\mu(P,t)} M(H,t), \tag{3}$$

where $\mu(H,t)$ is the average fitness of the structures or schemata that are in both $P(t)$ and in H, and where $\mu(P,t)$ is the average fitness of all structures in $P(t)$. Thus, the number of samples allocated to a hyperplane H changes exponentially over time, growing for the above average and dwindling for the below average. The relationship in equation (3) applies to each hyperplane represented in the population. In general, in a population of N binary structures of length L bits, between 2^L and $N2^L$ distinct hyperplanes are available for sampling. GAs test and search for these high-performance substrings or schemata, which far outnumber the individual structures in the population at any time, since a single structure is an instance of 2^L distinct hyperplanes ($\Sigma_{k=0}^{L}\,{}_LC_k$).

The most important recombination operator is *crossover*. Under the crossover operator, two structures in the mating pool exchange portions of their binary representation. This can be implemented by choosing a point on the structure at random—the crossover point—and exchanging the segments to the right of this point. For example, let two "parent" structures be

$$x_1 = 100{:}01010, \quad \text{and}$$
$$x_2 = 010{:}10100.$$

and suppose that the crossover point has been chosen as indicated. The resulting "offspring" structures would be

$$y_1 = 100{:}10100, \quad \text{and}$$
$$y_2 = 010{:}01010.$$

Crossover serves two complementary search functions. First, it provides new strings for further testing within the structures already present in the population. In the above example, both x_1 and y_1 are representatives of the structure or schema 100#####, where the # means "don't care, because the value at this position is irrelevant." (If 1001 is a point, then 100# is a line, and 10## is a plane, and 1### is a hyperplane.) Thus, by evaluating y_1, the GA gathers further information about this structure. Second, crossover introduces representatives of new structures into the population. In the above example, y_2 is a representative of the structure #1001###, which is not represented by either "parent." If this structure represents a high-performance area of the search space, the evaluation of y_2 will lead to further exploration in this part of the search space. The GENESIS package (Grefenstette 1987), which we use, implements two crossover points per mating.

A second operator is *mutation*: each bit in the structure has a chance of undergoing mutation, based on an interarrival interval between mutations. If mutation does occur, a random value is chosen from {0,1} for that bit. Mutation provides a mechanism for searching regions of the allele space not generated by selection and crossover, thus reducing the likelihood of local optima over time, but mutation is capable only of providing a random walk through the space of possible structures.

The basic concepts of GAs were developed by Holland (1992) and his students. Theoretical considerations concerning the allocation of trials to structures (Holland 1992, 1986; De Jong 1980) show that genetic techniques provide a near-optimal heuristic for information gathering in complex search spaces. A number of experimental studies (De Jong 1980; Bethke 1981; Grefenstette 1986) have shown that GAs exhibit impressive efficiency in practice. While classical gradient search techniques are more efficient for problems which satisfy tight constraints (e.g., continuity, low-dimensionality, unimodality, etc.), GAs consistently out-perform both gradient techniques and various forms of random search on more difficult (and more common) problems, such as optimisations involving discontinuous, noisy, high-dimensional, and multimodal objective functions (Holland 1984; Brady 1985).

GAs do not require well-behaved, convex objective functions—indeed, they do not require closed objective functions at all—which provides an opportunity for an exhaustive study of the solution to repeated games. This is possible because to use GAs to search for better solutions it is sufficient that each individual solution can be scored for its "evolutionary fitness:" in our case the aggregate score of a repeated game provides that measure, but in general any value that depends on the particular pattern of each individual string will do.

4. Results

THE purpose of bringing the powerful tool of GAs to bear on the problem of "breeding" strategies for the repeated Prisoner's Dilemma is twofold: by replicating the environment of the repeated two-person Prisoner's Dilemma computer tournaments to attempt to breed strategies equal to or better than those entered in previous tournaments, and to examine the structures of "good" strategies to learn whether there exist other "fit" traits embodied in such strategies as specific alleles. In particular, is there a strategy—encoded by a string such as those illustrated above—that systematically outperforms the triumphant Tit for Tat?

At the risk of tedium, we repeat that there is *no* best winning strategy in the non-zero-sum game of Prisoner's Dilemma: we can only speak of a best strategy given any particular environment or *niche*. We report on the results from six niches: (*a*) a niche of Always Defect, (*b*) a niche of Always Coöperate, (*c*) a niche of Tit for Tat, (*d*) the 5-rule niche described by Axelrod (1984), (*e*) the 8-rule niche described by Axelrod (1987), and (*f*) the 5-rule Axelrod niche but with "noise" added, to simulate Nalebuff's (1987) repeated Prisoner's Dilemma game with imperfect information—a 5% probability that what each believes the other to have done in the previous round is incorrect.

4.1 Breeding against "Always Defect"

When a random population of genetic strategies plays against Always Defect, those strings that embody—whose behaviour exhibits—"nastiness" will do better than strategies which coöperate despite the hostile environment. Thus, the scores (average and best) of the population rapidly converge, so that the best individual strategies match Always Defect, with an average score of 1 per round. Note that these strings will not resemble those of Table 5, since some states will never occur—those in which the strategy coöperates—and so the corresponding alleles will not be given a chance to test their fitness against the niche, and so cannot be selected. From Table 2, only states q = 2 and 3 are relevant, since the other states correspond to C by the other player, and hence are never seen; the optimal 1-round-memory string is ##11: defect against the other's Always Defect. From Table 8, the strings from selection against Always Defect should resemble those of Table 9.

```
                              ##11
                       ######## ##11##11
######## ######## ######## ######## ######## ##11##11 ######## ##11##11
```

TABLE 9. Optimal Strings against "Always Defect".

4.2 Breeding against "Always Coöperate"

When a random population of genetic strategies faces the strategy of Always Coöperate, those that embody "nastiness" to take advantage of the "niceness" of the niche will be selected for. These strings will not resemble those of Table 6, since that was for Always Coöperate on the part of the string, not the niche. If the niche is Always Coöperate, then only certain alleles will be tested for their fitness, those at positions on the string corresponding to the opponent's coöperating. The strings from selection against Always Coöperate should resemble those of Table 10.

4.3 Breeding against Tit for Tat

When faced with a Tit-for-Tat niche, the population of strings will soon resemble Tit for Tat at least in their fitness scores, since mutual coöperation is rewarded with higher "fitness" and

11##

11##11## ########

11##11## ######## 11##11## ######## ######## ######## ######## ########

TABLE 10. Optimal Strings against “Always Coöperate”.

defection is punished with lower “fitness.” The scores of the best individual strings will soon equal those of Tit for Tat, and the mean of the population will rise towards this level. The best response against a Tit for Tat strategy is to mimic it, which means coöperating (unless both are destructively defecting). It is sometimes overlooked that ease of recognition is one of the four traits advantaging Tit for Tat: the others are niceness (never first to *D*), swift to anger (Tit for Tat), and not grudge-bearing (no more than one tit per tat).

Except for 1-round memory, the strings from selection against Tit for Tat will only test a subset of the alleles, as seen in Table 11, since the niche of Tit for Tat will not provide an event history corresponding to the # symbol. Indeed, with genetic drift the alleles shown in the Figure—with the exception of $q = 0$, corresponding to continuing mutual coöperation—will eventually not be selected for against Tit for Tat. The next niche provides a greater range of behaviour and also a test of the alleles at positions other than $q = 0$.

0011

0#0#0#0# #1#1#1#1

0#0##### #0#0#### 0#0##### #0#0#### ####1#1# #####1#1 ####1#1# #####1#1

TABLE 11. Optimal Strings against “Tit for Tat”.

A GA simulation against Tit for Tat with 3-round memory resulted in a family of strategies that were all coöperating, even though convergence of structure had not occurred; that is, there was convergence of *behaviour* for which convergence of *structure* is sufficient but not obviously necessary.

4.4 Breeding against Axelrod's Five-Rule Niche

In his 1984 report, Axelrod described an environment of five of the rules entered in his second tournament that could be used as representatives of the complete set of 63 rules (or strategies) entered in the tournament, in the sense that the scores that a given strategy scores against them could be used to predict the average score the strategy would obtain over his full set of 63.

Each set of games of Axelrod's tournament was played for 151 rounds. He reported (1984, p. 199) the formula for the predicted tournament score in his second tournament as

$$\begin{aligned} T_5 = 120.0 &+ (0.202)\,S_6 + (0.198)\,S_{30} + (0.110)\,S_{35} \\ &+ (0.086)\,S_{27} + (0.072)\,S_{46}, \end{aligned} \tag{4}$$

where T_5 is the predicted tournament score (or fitness) of a strategy, and S_j is the score which that strategy obtains against the jth rule. Axelrod reported that the T_5 estimates correlate with the actual tournament scores at $r = 0.979$, and $r^2 = 0.96$. So 96% of the variance in his tournament scores is explained by knowing a rule's performance with only these five representative strategies or rules.

Rule S_6 (the subscript represents the ranking in his second tournament), known as *Tit for Tat with Check for Random*, was entered by Jim Graaskamp and Ken Katzen. Rule S_{30}, known as the *Revised State Transition*, was entered by Jonathan Pinckley. Rule S_{35}, known as *Discoverer*, was entered by Robert Adams. Rule S_{27}, known as *Tranquilizer*, was entered by Craig Feathers. Rule S_{46}, known as *Tester*, was entered by David Gladstein. Of course, there is nothing demonstrably optimal about the particular set of 63 rules entered in Axelrod's tournament, but, as he points out, by altering the weightings of the five representative rules, it is possible to construct hypothetical variants of the tournament. Indeed, Nachbar (1988) argues that the results from Axelrod's second tournament are tainted by the entrants' prior knowledge of the results of the first, which may have been suboptimal. Nonetheless, against the niche of Axelrod's 5-rule environment, as we see below, Tit for Tat is very successful. In the next section we examine results when breeding against an 8-rule niche (Axelrod 1987).

Before using Axelrod's 5-rule niche to "breed" genetic strategies, we calibrated three simple strategies against it, with the mean results shown in Table 12 (for 151 rounds). As expected, Always Defect results in a very low score, and Tit for Tat does much better than Always Coöperate.

Strategy	Score
Always Defect	342.780
Always Coöperate	406.146
Tit for Tat	422.446

TABLE 12. Simple Strategies against Axelrod's Five-Rule Niche.

Before breeding 3-round-memory strategies against the 5-rule niche, we bred a population of 50 1-round-memory strategies. After about 90 generations, the following structure converged: 010010, where the first four bits are the mapping from state (Table 2) to action, and the last two are the phantom memory (10 $\rightarrow$ *DC*). This is not Tit for Tat (001100), but something nastier, the first move of which is to defect (because of the specific phantom memory and the 1 corresponding to state 1). Figure 4 presents a Moore machine corresponding to 010010.

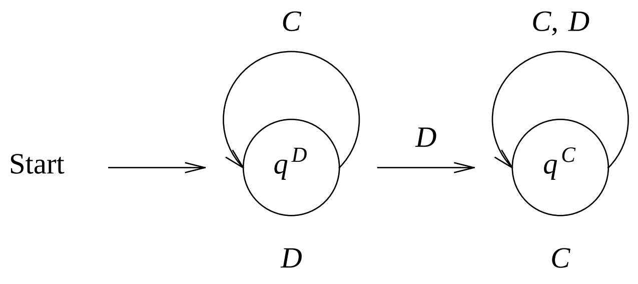

Figure 4. The "010010" Moore Machine

Comparison of this Figure with Figures 1 and 2 shows that 010010 is closer to Always Coöperate than to Tit for Tat, since once the string coöperates once, it does thenceforth, whatever the other player does, but that it starts off defecting, and continues to do so so long as the other player lets it, by not retaliating. Against Tit for Tat, the pattern is $\{DC, DD, CD, CC, CC, \cdots\}$, so that string 010010's undiscounted score is equal to Tit for Tat's. Its score averaged 447.273 against the 5-rule niche, against Tit for Tat's of 422.446. This string is an example of a "trigger" strategy: once its opponent first defects, it never itself defects again.

The population of 50 3-round-memory genetic strategies was bred for 2,000 generations, a total of 100,000 trials, each trial comprising 151 rounds of the iterated Prisoner's Dilemma against the five representative rules or strategies, a grand total of 75,500,000 rounds of the single Prisoner's Dilemma. This took 76 minutes on the AGSM's Pyramid 90X, as sole user.

The results were similar to Axelrod's (1987). The best individual scored 475.91 which was greater than the mean of 422.446 for Tit for Tat. (Since strategies S_{27} and S_{35} are non-deterministic, the scores of their competitors may vary by up to 20%, as Axelrod reports.) The uncertainty associated with this environment, although in a sense adding to the work necessary to evaluate the robustness of strategies, does provide a more varied environment against which to select for alleles. The current best individual appeared after trial 15,500 in the 309th generation.

The family of highest-scoring strings is shown in Table 13, against the necessary Tit for Tat string from Table 7, which provides a basis for comparison, at least in the sixteen positions that characterise Tit for Tat.

```
0#00#### ###00#0# ##0##11# #1##01## 0###1#1# #####0## #####00# #0###### #11#01
00####00 ######## ######## 00####00 11####11 ######## ######## 11####11 000000
```

TABLE 13. The Best Family String, Bred against Axelrod's Five-Rule Niche.

Examination shows that there is very little fit between the best family and Tit for Tat: only in two positions (q = 0 and 46) plus one bit of phantom memory do they agree, and they explicitly disagree in three positions (q = 33, 40, and 57) and in three bits of phantom memory. Thus there is virtually no comparison between the best family and Tit for Tat.

Note that the score of the best 3-round-memory strategies of 475.91 is higher than that of the best 1-round-memory strategy of 447.273, strongly suggesting that the increased level of strategic complexity allowed by the longer bit-strings of the 3-round-memory strategies is rewarded by higher scores in the repeated Prisoner's Dilemma. If true, this provides a research programme—to answer the first of the two questions at the end of Section 2.1: Given any level of strategic complexity, what is the most successful strategy in competing against a given niche of strategies? The second question—With no limit on strategic complexity, can Tit for Tat be soundly bettered?—seems to have been clearly answered in the positive.

4.5 Breeding against Axelrod's Eight-Rule Niche

In his 1987 paper, Axelrod spoke of an environment of eight of the rules entered in his second tournament, which he reported as having an r^2 of 0.98. In a personal communication to the author, Axelrod revealed the formula for his 8-rule niche as

$$\begin{aligned} T_8 = 110.55 &+ (0.1574)\,S_{30} + (0.1506)\,S_6 + (0.1185)\,S_{35} \\ &+ (0.0876)\,S_{27} + (0.0579)\,S_{55} + (0.0492)\,S_{34} \\ &+ (0.0487)\,S_{46} + (0.0463)\,S_{16}, \end{aligned} \tag{5}$$

where T_8 is the predicted tournament score of a strategy, and S_j is the score which that strategy obtains against the *j*th rule. The three additional rules are: rule S_{55}, *Adjuster*, entered by Scott Feld; rule S_{34}, *Slow-Never*, entered by Roger Falk and James Langsted; and rule S_{16}, *Fink*, entered by Richard Hufford.

As before, we calibrated three simple strategies against the 8-rule niche, with the mean results for 151 rounds shown in Table 14.

Strategy	Score
Always Defect	319.831
Always Coöperate	398.513
Tit for Tat	427.198

TABLE 14. Simple Strategies against Axelrod's Eight-Rule Niche.

The results of breeding strategies against the niche indicate a robustness for Tit for Tat not evident in the earlier results. With 1-round-memory strategies (strings or 4 bits, plus 2 for phantom memory), from a population of random strings, the string 001100 emerged as the highest scoring and sole survivor after 2400 trials. But from Table 2 we see that this is Tit for Tat, with the phantom memory of mutual coöperation.

With 2-round-memory strategies (strings of 16 bits, plus 4 for phantom memory), from a population of 50 random strings, the family of strings shown in Table 15 emerges. After 17,000 trials (339 generations) there was virtual convergence: 49 defected with state 2, one coöperated. The average score of the family was 427.6, slightly above the score for Tit for Tat, again suggesting that higher levels of strategic complexity can result in more highly scoring strategies, although this cannot be pushed too far in this case, given the stochastic nature of the niche. (Two of the rules are stochastic, as in the 5-rule niche.) Table 15 also presents the necessary string for Tit for Tat from Table 7. We see that the necessary bits agree. What of sufficiency? For Tit-for-Tat behaviour it would be sufficient that the first eight states mapped to 0 (*C*) and the second eight to 1 (*D*), with a phantom memory of mutual coöperation.

```
00#00100 11101111 0000
00####00 11####11 0000
```

TABLE 15. Two-Round-Memory Strategy and Tit for Tat

We see that states 5 and 11 of the family are the opposite of those sufficient for Tit for Tat behaviour, but these states may never be encountered, given the others. For 1- and 2-round-memory strategies against Axelrod's 8-rule niche, Tit for Tat seems good.

For 3-round-memory strategies there is again a family of strings that do converge (but for two bits of the 70-bit string: 64 bits of strategy, plus 6 bits of phantom memory). The best family string is shown in Table 16, together with the necessary string for Tit for Tat, from Table 7.

```
00101100 01100010 01000100 #010000# 11101110 10000101 01110100 10001011 100000
00####00 ######## ######## 00####00 11####11 ######## ######## 11####11 000000
```

TABLE 16. The Best Family String, Bred against Axelrod's Eight-Rule Niche.

Inspection shows that there are disagreements between the best string and necessary Tit for Tat in two states (q = 39 and 57) and one bit of phantom memory; in two other positions (q = 24 and 31)—the only two positions where convergence had not occurred after 12,000 trials (generation 239)—the string is ambiguous: 49 of the 50 individuals in the 239th generation had a 0 at position 24, which agrees with the necessary Tit for Tat, and another 49 had a 1 at position 31, which does not agree. Nonetheless, the best family string is close to Tit for Tat in structure and score (at 426.29, almost identical to Tit for Tat).

4.6 Breeding Strategies in a Noisy Game

Following Nalebuff's challenge (1987), we have attempted to breed strategies in an environment with "noise," specifically in Axelrod's 5-rule niche where there is a 5% probability that any player misreads his opponent's previous move, and believes it was *C* when it was *D*, or vice versa. Strings will be selected that do *not* engage in "a mindless and almost slapstick routine of punishing the other player: player *A* punished *B* for defecting and then *B* punishes *A* for punishing him and so on" (Nalebuff 1987, p. 187): this is the situation which may occur when there is noise and Tit-for-Tat-ish competitors. The first results don't seem to be generalisable, and the best of the noisy strategies so far is shown in Table 17.

```
00000001 00011000 11011101 10000111 01001000 10111011 10000101 10110011 010101
00####00 ######## ######## 00####00 11####11 ######## ######## 11####11 000000
```

TABLE 17. The "Noisy" String and Tit for Tat.

We see that in positions 0, 1, 6, 25, 33, 50, 56, and 57, it behaves as does Tit for Tat, but that at the eight other Tit-for-Tat-consistent positions (7, 24, 30, 31, 32, 38, 39, and 51) its behaviour is opposite that of Tit for Tat. Its score was 400.09, which is difficult to judge as good or not, given the difficulty of calibration with the requirement that both players face noisy reporting of the other's moves. It is possible that a strategy which looked at the cumulative score, as well as previous moves, would do well in this environment, but, before modelling such behaviour in a genetic-algorithm context, we must squeeze all possibilities from the purely move-regarding strategies. It may be that 151 rounds of the repeated game do not provide us with a test of sufficient power to feel confident that the schemata or sub-structures are being adequately selected for or against. For any given level of noise or uncertainty in the one-shot game, a larger number of rounds will provide greater confidence in structures so bred.

5. Conclusions

THERE have been few other attempts to harness the power of the GA to search the enormous number of possible strategies in the repeated two-person Prisoner's Dilemma (Holland and Miller 1991). Axelrod (1987) pioneered 3-round-memory strategies, and Miller (1989) searched explicitly for finite automata to play the repeated game. Initial results look interesting. With a reasonable length of simulation we have been able to "breed" a 3-round-memory strategy which is almost always better than Tit for Tat in 5-rule niche that approximates Axelrod's second computer tournament. Although the results from the 8-rule niche are not so suggestive, it appears that the answer to the question posed at the end of Section 2.1 is that allowing a greater level of strategic complexity does result in strategies which can outperform Tit for Tat. If this is so, then there is a research programme to answer the other question posed at the end of Section 2.1: at any given level of strategic complexity, what is the most successful strategy in competing against a given niche of rules?

In a survey of developments in the evolution of coöperation, Axelrod and Dion (1988) spend some time discussing the ways in which noisy environments can influence such development, from a theoretical perspective. Miller (1989) has systematically bred finite automata in noisy environments of various kinds, including that described by Nalebuff (1987). The two questions posed at the end of Section 2.1 above remain to be answered for such noisy environments with increasingly complex strategies, as reflected in ever longer memory.

In the framework above, future work will examine the environment more closely, and seed it with successful strings previously bred. An extreme version of this is "coevolution": pitting each individual string in a population not against some unchanging environment, as we have done above, but against every other member of its generation exhaustively (Miller 1989, Marks 1992*b*[5]). For any but the smallest populations such a programme would be extremely CPU-intensive, but its outcome can be predicted: in-bred strings that coöperate against each other—and thus do well in their mutually rewarding environment—but that might be easy pickings for any invading "nasty" strategy. It is difficult to imagine that resistance to nastiness would remain in a highly in-bred population of strategies except by chance, since such a trait would not be selected for after the first few score generations of genetic drift.

A market of three sellers facing an elastic demand and selling a homogeneous output can be modelled as a three-person PD: each seller's profits would be maximised by a coöperative, high price (the Pareto price), but competition drives the price down towards the Nash price and hence each seller's total profits down, even though with elastic demand the market grows. Following a

5. Marks (1992*b*) referred to it as "bootstrapping"

similar tournament at MIT (Fader and Hauser 1988), the author ran a computer tournament at the AGSM to see whether entrants' strategies could generalise Tit for Tat to the three-person game. In a second tournament at MIT, the author's strategy won. It is difficult to conclude that it would always win, or indeed that any strategy is the "best," since performance depends on the population of competing strategies. Marks (1992*b*) reports efforts to use the GA to breed winning strategies in three-person games.

A final comment should draw attention to a criticism of the Axelrod tournaments (and hence to other such tournaments) by Nachbar (1988), who argued that the robustness of Tit for Tat in Axelrod's second strategy was conditioned by the announcement of its success in the first tournament. The strings bred against a niche environment are not conditioned by prior knowledge, and so in a sense the work reported here answers Nachbar's criticisms. But the specific niche was, of course, derived from Axelrod's second tournament, which was perhaps conditioned by the expectations of the entrants after the first tournament's results were announced. Only the coevolution or bootstrapping mentioned above will clearly avoid Nachbar's criticisms.

Acknowledgements:

The author would like to thank Robert Axelrod for his courteous and prompt response to queries, and John Grefenstette for his algorithm and advice. Gerry Garvey, Chris Adam, David Midgley, Luigi Ermini, and David Simmonds have provided helpful comments. Greg James helped with the debugging. The Australian Graduate School of Management, the Australian Research Council, and the Graduate School of Business at Stanford University provided research support.

References:

Aumann R., 1985, Repeated games, in *Issues in Contemporary Microeconomics and Welfare,* ed. by G.R. Feiwel, London: Macmillan, 209–242.

Axelrod R., 1984, *The Evolution of Coöperation,* N.Y.: Basic Books.

Axelrod R., 1987, The evolution of strategies in the iterated Prisoner's Dilemma, in *Genetic Algorithms and Simulated Annealing,* L. Davis (ed.), London: Pittman.

Axelrod R. and Dion D., 1988, The further evolution of coöperation, *Science,* **242**: 1,385–1,390, 9 Dec.

Ben-Porath E., 1987, Repeated games with finite automata, Stanford University Institute for Mathematical Studies in the Social Sciences, Tech. Report No. 515, August.

Bethke A.D., 1981, Genetic algorithms as function optimizers, Univ. of Michigan Ph.D. thesis, Univ. Micro Films, 81–06101.

Brady R.M., 1985, Optimization strategies gleaned from biological evolution, *Nature,* **317**: 804–806, 31 Oct.

Cohen M.D., and Axelrod R., 1984, Coping with complexity: the adaptive value of changing utility, *Amer. Econ. Rev.,* **74**: 30–42.

De Jong, K.A., 1980, Adaptive system design: a genetic approach, *IEEE Trans. on Systems, Man, and Cybernetics,* SMC–**10**: 566–524.

De Jong, K.A., 1990, Genetic-algorithm-based learning, in *Machine Learning: An Artificial Intelligence Approach,* Volume 3, ed. by Y. Kodratoff and R.S. Michalski, San Mateo, Calif.: Morgan Kaufmann, 611–638.

Dupré J., (ed.), 1987, *The Latest on the Best: Essays on Evolution and Optimality,* Camb., Mass.: M.I.T. Press.

Fader P.S., and Hauser J.R., 1988, Implicit coalitions in a generalised Prisoner’s Dilemma, *J. Conflict Resol.,* **32**: 553–582.

Futia C., 1977, The complexity of economic decision rules, *J. Math. Econ.,* **4**: 289–299.

Goldberg D.E., 1988, *Genetic Algorithms in Search, Optimization, and Machine Learning,* Reading, Mass.: Addison-Wesley.

Gottinger H.W., 1983, *Coping with Complexity: Perspectives for Economics, Management and Social Sciences,* Dordrecht: D. Reidel.

Gottinger H.W., 1987, Choice and complexity, *Math. Soc. Sciences,* **14**: 1–17.

Grefenstette J.J., 1986, Optimization of control parameters for genetic algorithms, *IEEE Trans. on Systems, Man, & Cybernetics,* SMC–**16**: 122–128.

Grefenstette J.J., 1987, A user’s guide to GENESIS, mimeo., Washington D.C.: Navy Center for Applied Research in AI, Naval Research Lab., August.

Hofstadter D.R., 1985, Dilemmas for super-rational thinkers, leading up to a Luring Lottery, in his *Metamagical Themas: Questing for the Essence of Mind and Pattern,* N.Y.: Basic Books.

Holland J.H., 1984, Genetic algorithms and adaptation, in *Adaptive Control of Ill-Defined Systems,* ed. by O. Selfridge, E. Rissland, and M.A. Arbib, N.Y.: Plenum, 317–333.

Holland J.H., 1992, *Adaptation in Natural and Artificial Systems,* Cambridge: MIT Press, (1975), 2nd. edition.

Holland J.H., and Miller J.H., 1991, Artificially adaptive agents in economic theory, *Amer. Econ. Rev.,* **81**(2): 365–370.

Luce R.D. and Raiffa H., 1957, *Games and Decisions: Introduction and Critical Survey,* New York: Wiley.

Marks R.E., 1992*a*, Repeated games and finite automata, in *Recent Developments in Game Theory,* ed. by J Creedy, J. Borland, and J. Eichberger, Aldershot: Edward Elgar, 43–64.

Marks R.E., 1992*b*, Breeding hybrid strategies: optimal behaviour for oligopolists, *Journal of Evolutionary Economics,* **2**: 17–38.

Miller J.H., 1989, The coevolution of automata in the repeated Prisoner's Dilemma, Santa Fe Institute Economics Research Program Working Paper 89-003.

Moore E.F., 1956, Gedanken experiments on sequential machines, in *Automata Studies,* ed. by C.E. Shannon and J. McCarthy, Princeton: Princeton Univ. Press, 129–153.

Nachbar J.H., 1988, The evolution of coöperation revisited, mimeo., Santa Monica: RAND Corp., June.

Nalebuff B., 1987, Economic puzzles: noisy prisoners, Manhattan locations, and more, *J. Econ. Persp.,* **1**: 185–191.

Nelson R.R, and Winter S.G., 1982, *An Evolutionary Theory of Economic Change,* Cambridge: Belknap Press of Harvard University Press.

Neyman A., 1985, Bounded complexity justifies coöperation in the finitely repeated Prisoners' Dilemma, *Econ. Lett.,* **19**: 227–229.

Radner R., 1986, Can bounded rationality resolve the Prisoners' Dilemma? in *Contributions to Mathematical Economics in Honor of Gérard Debreu,* ed. by W. Hildenbrand and A. Mas-Colell, Amsterdam: North-Holland, 387–399.

Rubinstein A., 1986, Finite automata play the repeated Prisoners' Dilemma, *J. Econ. Theory,* **39**: 83–96.

Schaffer J.D. and Grefenstette J.J., 1988, A critical review of genetic algorithms, mimeo.

Simon H.A., 1972, Theories of bounded rationality, in *Decision and Organization,* ed. by C.B McGuire and R. Radner, Amsterdam: North Holland, 161–188.

Simon H.A., 1978, On how to decide what to do, *Bell J. Econ.,* **9**: 494–507.

Ulph A., 1987, Recent advances in oligopoly theory from a game theory perspective, *J. Econ. Surveys,* **1**: 149–172.