J. V. HOWARD

COOPERATION IN THE PRISONER'S DILEMMA

1. INTRODUCTION

The payoff matrix below is an example of the Prisoner's Dilemma (PD) game. If both players cooperate (C) they each get 2: if both defect (D) they each get 1. However if one cooperates but not the other, the cooperator gets 0 whilst the defector gets 3.

		Column	
		C	D
Row	C	2,2	0,3
	D	3,0	1,1

There are three arguments for playing D in this game. Firstly, (D,D) is the only Nash equilibrium: for any other pair of pure or mixed strategies at least one player has an incentive to change his strategy (given the other's strategy is fixed). Secondly, it is the maximin strategy: D guarantees at least 1, C might yield 0. Thirdly, and most powerfully, D strictly dominates C: whatever strategy Column chooses, Row is at least one unit better off if he plays D.

Similar results hold if the game is repeated a fixed number of times. All the Nash equilibria imply that both players play D in every game. Pairs of maximin strategies lead to the same result. And successive elimination of dominated strategies (equivalent to 'reducing the game backwards') also produces a string of defections. (However cooperative outcomes can be Nash equilibria if the number of repetitions of the game is uncertain.)

Nonetheless arguments have been made in favour of playing C even in a single play of the PD. The one that interests us relies heavily on the usual assumption that both players are completely rational and know everything there is to know about the situation [1]. (So for instance, Row knows that Column is rational, and Column knows that he knows it, and so on.) It can then be argued by Row that Column is an individual very similar to himself and in the same situation as himself. Hence whatever he eventually decides to do, Column will also necessarily do the same (just as two good students given the same sum to calculate will necessarily arrive at the same answer). Hence if Row chooses D, so will Column, and each will get 1. However if Row chooses C, so will Column, and each will then get 2. Hence Row should choose C.

This is clearly an unusual argument! It was put forward by Rapoport in 1966, and discussed by Davis in 1977 [2]. (However, Rapoport did not press the argument strongly, and later supported another resolution of the problem, namely Nigel Howard's metagame analysis (1971) [3].) A recent statement of the argument is Hofstadter (1983) [4].

The idea of behaving so as to necessitate that another individual act in a certain way is not unique to the PD. Ayer (1956) gives the example of a Calvinist who abstains from sin in order to have been saved [5]. In Newcomb's Paradox it can be argued that the Chooser should open only one box so as to necessitate that the Predictor shall have previously placed one million pounds in it. (See the discussion in Kavka (1980) [6].) Again, it seems irrational to bother to vote in an election which almost certainly will not be won or lost by one vote. However, if your action necessitates that a large group of citizens with similar predispositions will (or have) also voted, it would be rational.

We do not propose in this paper to dive into the subtleties of this sort of argument (including the murky waters of free will, determinism and backwards causation). Instead we propose to work with a 'model' of the PD. The model will have computer programs instead of individuals, and there will be no direct assumption of rationality (and so no assumption that there always is a rational choice of strategy).

Replacing individuals by programs in real or imaginary situations seems to us often to make a philosophical problem much clearer, and to illuminate some of its essential features. We used this approach for the problem of statistical inference in Howard (1975) [7]. The idea of having games played by programs instead of people was introduced by Axelrod (1980) and discussed in Axelrod and Hamilton (1981) [8]. The programs were enlisted in two rounds of a 'tournament', and played each other – including a clone of themselves – 200 times in the first round and a random number of times (with a median of 200) in the second round. They knew whether they were playing the same opponent (so they could 'recognize' individuals), and they could remember (if they wished) the results of previous games against the same player. Thus the tournaments simulated the repeated PD.

In Axelrod's tournaments Rapoport's strategy TIT-FOR-TAT (cooperate in the first game against an unknown opponent, thereafter copy his previous move) did amazingly well. It won almost all the tournaments (in the sense of gaining the highest total payoff), despite the fact that by its nature it can never win any individual contest. TIT-FOR-TAT can be regarded as one solution to the repeated PD, in the sense that a population of TIT-FOR-TAT programs always obtains a Pareto optimal payoff (one that cannot be improved for both players at the same time), and is also evolutionarily stable according to the definition of Maynard Smith and Price (1973), and Maynard Smith (1982) [9]. This means that any small invading strategy (program) will be less successful than the bulk of the population, and will die out in the course of evolution (assuming the payoffs of the game are in units of Darwinian fitness and the programs breed true).

In this sense, then, Axelrod's tournaments suggest a solution to the repeated PD. We shall consider an imaginary tournament with different rules which will model the single-shot PD. In this model the strange argument of Rapoport and others for playing cooperatively in such games becomes completely clear, and in fact becomes a possible solution to the single shot PD, in just the same sense as TIT-FOR-TAT is for the repeated PD.

2. RECOGNITION

In the Axelrod tournaments, the competitors could *recognise* each other as individuals. They knew how many times they had played a particular opponent before, and if they wished they could record and recall the result of each previous encounter with this opponent. In the Rapoport argument it is also crucial that the player *recognise* that his opponent is also rational (and in fact is very similar to himself).

However the dominance argument does not require that the opponent be rational. It suggests that in a single-shot game it is always best to play non-cooperatively. Most players in Hofstadter's tournament seemed to accept this argument, and hence defected.

We now wish to describe an imaginary tournament in which the players never recognise each other as individuals. (This might be because they never in fact meet the same opponent twice, or because they are incapable of recognising someone when they meet him again. The opponents are masked.) However the players can perceive the way their opponents think. They can do this because they are given as data the computer program of their opponent. Hence although players cannot recognise individuals, they can recognise *types*. (Two players with the same program are said to be of the same type.) Several copies of each program submitted would be entered into the tournament, and then each player would play every other player exactly once. The idea of recognition of types is due to K. G. Binmore, who suggested studying games between players who have their Gödel numbers written on their foreheads.

Players could of course remember (if they wished) the results of previous games against the various types of opponent. If they did this it would alter their own type, because we shall regard any stored data used by a program as being part of that program.

At first glance, it appears easy to write a program to do well in this tournament. We arrange that our program will read in the opponent's program, then simulate what it will do when fed with our program as data. Having decided what our opponent is going to do, we can then decide what our best strategy will be.

However, if the other program is trying to do the same sort of thing, we fall into an infinite regress, and both programs will compute indefinitely. In fact, to make the rules precise, we will have to insist that each program must produce an answer within a specified time, or a specified number of program steps.

On seconds thoughts, however, it seems unnecessary to try to find out what our opponent will do. Whatever he does, we do best by playing D, so why not just play D. (This is just the dominance argument.) Let us suppose that one of the program types embodies this non-cooperative strategy. We shall now show that there is a program that will do better than this strategy when matched against it.

Our contender will be a program that recognises its own type, (i.e. it recognises itself). When it recognises that its opponent is identical to itself, it plays C. Otherwise, it plays D. We shall call this the MIRROR strategy. Clearly such a program will do better than the non-cooperative strategy. Suppose for example that in the tournament are five copies of the non-cooperative strategy and five of the MIRROR strategy. Then the non-cooperative players will obtain 9 points, while the MIRROR players will obtain 13.

It is of course necessary to show that there can be a program that recognises itself. This is fairly easy to do using the standard methods of mathematical logic.[1] We first give the essential idea in the form of an algorithm in English which recognises itself.

Algorithm

Read the proposed algorithm and check that the first part of it is the text enclosed in quotation marks at the end of these instructions. Check that the remainder of it consists of a copy of the first part enclosed in quotation marks. If the proposed algorithm is of this form, declare that it is the same algorithm, otherwise declare that it is a different algorithm. The text referred to above is as follows. "Read the proposed algorithm and check that the first part of it is the text enclosed in quotation marks at the end of these instructions. Check that the remainder of it consists of a copy of the first part enclosed in quotation marks. If the proposed algorithm is of this form, declare that it is the same algorithm, otherwise declare that it is a different algorithm. The text referred to above is as follows."

The idea embodied in the algorithm can equally well be incorporated into a computer program. The Appendix gives such a program (written in a dialect of BASIC). It reads another BASIC program (terminated by "END") and prints "Same Program" or "Different Program" as appropriate. It is interesting that one of the programs (submitted by J. Graaskamp) enlisted in Axelrod's tournament tried to recognise its twin and modify its play accordingly. In this case the recognition was based on the way the other program played the first 56 moves.

Hence we have shown that under certain assumptions it can be sensible to play cooperatively even in a single-shot play of the Prisoner's Dilemma. One situation where it is sensible is when you can recognise that your opponent is similar to yourself. This is essentially the argument put forward by Rapoport. Another possible situation is outlined in the next section.

3. PROBLEMS AND EXTENSIONS

There will be many different programs that recognise themselves – for example copy the program in the Appendix, but insert some extra spaces in one of the lines (and alter the corresponding DATA statement). We then get a population of individuals of a number of different types who cooperate within a type but not across types, although the differences between types are minute. (This is scarcely unknown in the real world.) It does seem a genuine problem how to cooperate with a wider range of opponents whilst not laying oneself open to exploitation by people who do not regard you as one of their type (or class, race, species, etc.).

It might be objected that if the two players have the same program they are effectively the same individual, and there is really no proper play of the PD game at all. However it is easy to imagine more sophisticated versions of the self-recognition program which would examine the other program to see which subroutine it uses when confronted with the PD game. If this subroutine is identical to its own subroutine for the game it would cooperate, otherwise not. Then the two programs could be very different except for their approach to the PD game, but they would still recognise each other and cooperate. They would have the same 'gene' for playing PD. The programs could even keep a record of previous games and this would not matter as long as they did not refer to this record when playing the PD. These twin programs must therefore be regarded as separate individuals, just as twins are in reality.

Nor do we have a case of cooperation between related individuals, sharing some of the same genes. For in that situation cooperation occurs because the payoffs do not take the form of the PD table: in fact there is an identity of interest for the common genes, as if the original payoffs for the two players had been aggregated to give the following table.

		Column	
		C	D
Row	C	4,4	3,3
	D	3,3	2,2

In that case, there is no problem.

In fact, we could easily modify our algorithm to produce two programs, P and Q, such that P recognised Q and played cooperatively against it, and vice versa, but neither P nor Q recognised itself. This would give a model of symbiosis. There would then be no question of the individuals' being related. Whether cooperation ever occurs in nature by this mechanism (recognition of genetic code) we are not able to say. However, evolutionary biologists have considered the idea that a gene might have two distinct effects: firstly to produce a conspicuous label in individuals carrying the gene, and secondly to confer a tendency to behave altruistically towards bearers of the label. This is known as the 'green-beard effect' [10]. It is clearly vulnerable to mutants which bear the label (the green-beard) without the altruism. Our model is the extreme case where the 'label' is the gene itself, and hence not vulnerable to mutants. It seems doubtful whether any examples of the green-beard effect exist in Nature, and even less likely that direct recognition of sections of genetic code ever occurs.

Other interesting programs can be envisaged. For example, suppose a fixed time is allowed for computation before each play of a game. We can imagine 'superior' programs which run on a fast computer and 'inferior' ones which run on a slow computer. Suppose a superior program always has time to calculate how an inferior program will play against it. Then a possible strategy for a superior program would be always to play what it forecasts the inferior program will play. (Of course its forecast is always correct.) We shall call this strategy TAT-FOR-TAT. It is a first-level metagame strategy in the sense of Nigel Howard (1971) [11]. Inferior programs faced with this superior strategy are then effectively confronted with Newcomb's problem. Despite the dominance argument, it is best for an inferior program to cooperate when confronted with the TAT-FOR-TAT oracle.

Imagine, for example, that we are writing a program for a slow computer. We know there will be one superior type of program (playing TAT-FOR-TAT) and we have a listing of it. We write our program so that it first checks if it is playing the superior program (by storing a copy of the listing as data within our program). If it is, then we arrange for it to cooperate. If it is not, we check to see if it is playing a copy of itself. Again, if it is, we cooperate. Otherwise our program will play D. If several people write programs with this general strategy, we will finish with a population consisting of one type of superior program and several types of inferior program. An inferior program cooperates with a superior program and with programs of its own type, but not with inferior programs of different types. However a superior program elicits cooperation from all opponents (assuming it is self-recognising). In this population the superior programs do best.

Of course we can easily envisage even faster computers giving rise to extrasuperior programs. Alternatively, instead of insisting that a program computes for not more than some fixed time, we might simply insist that it must always stop eventually. In this case, a program could try to find what its opponent would do by simulating N of its program steps. If the opponents program did not halt within the N steps, it would (say) play D, otherwise it would play TAT-FOR-TAT. In this way we would get a hierarchy of programs (with larger and larger values for N), so that no matter how complicated a program was, there would always be another program that could forecast its moves. There would, however, be no godlike program superior to all others.

4. DISCUSSION AND CONCLUSIONS

Biologists have suggested that individuals may cooperate in situations of potential conflict in two cases:

(a) if they recognise that the other player is a close relative, or

b) if they recognise the other player as someone with whom they are playing a sequence of games.

Both situations depend on recognising something about the other player. And a central assumption of much of Game Theory is that each player knows that the other players are rational (and they know that he knows, etc.). This assumption is not clearly specified unless "rational" has previously been defined. So we can regard Game Theory as attempting to provide a satisfactory definition of rational behaviour in game playing. The attempt is successful in the case of zero-sum games, but not for non-zero-sum games. The original problem must therefore be specified more clearly. We have tried to explore the use of Binmore's idea, replacing the assumption that players know each other to be rational by the assumption that they know each other's game algorithms. So here is a third way in which individuals may recognise each other. (We have shown that this does not necessarily mean that they can forecast what pure or mixed strategies the other players will use.)

In general a good algorithm (program) to use will depend on the opponents' programs. As in the Axelrod tournaments there may not be a best program for all populations of opponents. However, two points should be noted. Firstly, we are closer to the pure Game Theory problem of the single-shot game: in our tournaments there need be no memory of previous encounters. Secondly, we can define an ideal population for playing the Prisoner's Dilemma. If all players use the self-recognition program listed in the Appendix, and play cooperatively only if they recognise their opponents as their twins, then every game will be played cooperatively. Moreover no small group of invading programs could exploit the existing population. In fact, the invaders would do badly and die off after a few generations of competitive interaction. In this sense then, in the new formulation the Prisoner's Dilemma has a solution, and that solution is very reminiscent of the argument given by Rapoport and others.

APPENDIX: A BASIC PROGRAM THAT RECOGNISES ITSELF

```
10 linenum = 0
20 FOR part = 1 TO 2
30 IF part = 1 THEN insert$ = " "
40 IF part = 2 THEN insert$ = "DATA"
50 RESTORE
60 REPEAT
70 linenum = linenum + 10
80 INPUT yourline$: READ endline$
90 myline$ = STR$(linenum) + insert$ + endline$
```

```
100 IF yourline$ <> myline$ THEN PRINT "Different Program":
    STOP
110 UNTIL endline$ = "STOP"
120 NEXT part
130 linenum = linenum + 10
140 INPUT yourline$
150 myline$ = STR$(linenum) + "END"
160 IF yourline$ <> myline$ THEN PRINT "Different Program":
    STOP
170 PRINT "Same Program"
180 STOP
190 DATA linenum = 0
200 DATA FOR part = 1 TO 2
210 DATA IF part = 1 THEN insert$ = " "
220 DATA IF part = 2 THEN insert$ = "DATA"
230 DATA RESTORE
240 DATA REPEAT
250 DATA linenum = linenum + 10
260 DATA INPUT yourline$: READ endline$
270 DATA myline$ = STR$(linenum) + insert$ + endline$
280 DATA IF yourline$ <> myline$ THEN PRINT "Different Pro-
    gram": STOP
290 DATA UNTIL endline$ = "STOP"
300 DATA NEXT part
310 DATA linenum = linenum + 10
320 DATA INPUT yourline$
330 DATA myline$ = STR$(linenum) + "END"
340 DATA IF yourline$ <> myline$ THEN PRINT "Different Pro-
    gram": STOP
350 DATA PRINT "Same Program"
360 DATA STOP
370 END
```

Notes

(i) Leading blanks are stripped from DATA strings;

(ii) STR$ is a function which converts a number into the string of digits which denote it.

NOTE

[1] As the paper was about to be printed, I discovered that this result, and its application to the PD, were established independently by R. P. McAfee of the University of Western Ontario in an unpublished article 'Effective Computability in Economic Decisions' (1984).

REFERENCES

[1] Anatol Rapoport: 1966, *Two-Person Game Theory* (Ann Arbor), pp. 139–141.

[2] A. Rapoport, *op. cit.*, pp. 141f; Lawrence H. Daviš: 1977, 'Prisoners, Paradox, and Rationality', *American Philosophical Quarterly* **14** (October), pp. 319–327.

[3] Nigel Howard: 1971, *Paradoxes of Rationality: Theory of Metagames and Political Behavior* (MIT Press); Anatol Rapoport: 1967, 'Escape from Paradox', *Scientific American*, July, pp. 50–56.

[4] Douglas R. Hofstadter: 1983, 'Metamagical Themes', *Scientific American*, June, pp. 14–18.

[5] A. J. Ayer: 1956, *The Problem of Knowledge* (Penguin Books), pp. 173–175.

[6] Gregory S. Kavka: 1980, 'What Is Newcomb's Problem About?', *American Philosophical Quarterly* **17** (October), pp. 271–280.

[7] J. V. Howard: 1975, 'Computable Explanations', *Zeitschr. f. math. Logik und Grundlagen d. Math.* **21**, pp. 215–224.

[8] R. Axelrod: 1980, 'Effective Choice in the Prisoner's Dilemma', *J. of Conflict Resolution* **24** (March), pp. 3–25. 'More Effective Choice in the Prisoner's Dilemma', *J. of Conflict Resolution* **24** (September), pp. 379–403. R. Axelrod and W. D. Hamilton: 1981, 'The Evolution of Cooperation', *Science* **211**, pp. 1390–1396.

[9] J. Maynard Smith and G. R. Price: 1973, 'The Logic of Animal Conflict', *Nature* **246**, pp. 15–18; J. Maynard Smith: 1982, *Evolution and the Theory of Games* (Cambridge University Press).

[10] W. D. Hamilton: 1964, 'The Genetical Evolution of Social Behaviour II', *Journal of Theoretical Biology* **7**, pp. 17–32; R. Dawkins: 1976, *The Selfish Gene* (Oxford University Press).

[11] N. Howard, *op. cit.*

London School of Economics,
Houghton Street,
London WC2A 2AE,
England