

Genetic and evolutionary approaches to the iterated prisoner's dilemma

Jordan W. Suchow

May 1, 2009

Two men are taken into custody by the police, because they are suspected conspirators in a murder case. At the police station, the men are separated into solitary-confinement jail cells, unable to communicate with one another. A police arbitrator approaches each suspect individually, and makes the following offer: turn in his conspirator and walk away scot-free (defect), or feign innocence (cooperate).

If both cooperate, the police will be foiled and the men will be kept for only a few months. If one defects while the other cooperates, the defector will be set free but the cooperator will receive a life sentence. If both defect, they will each receive a moderate 5-year sentence.

What should the conspirators do?

Assume that the prisoners are logical, and that each prisoner aims only to minimize his jail-time. Then each prisoner considers the situation in this logical chain of thought: "If the other conspirator were to cooperate, then I should defect so as to receive no punishment. If the other conspirator were to defect, then I should also defect so as to receive a lesser punishment." In either case, for both conspirators, the logical choice is to defect. But when both conspirators defect, the equilibrium is a suboptimal solution.

Herein lies the prisoners dilemma (Poundstone, 1993).

What is meant by a suboptimal solution? One measure of optimality is Pareto Optimality, which defines a "suboptimal" solution as one to which there exists an alternative that benefits at least one player, while hurting none (Pareto, 1906). Thus, the equilibrium solution to the prisoner's dilemma is suboptimal because if the two conspirators were to cooperate rather than defect, both players would benefit, while hurting none.

Traditionally, the prisoner's dilemma refers to isolated instances of the game, each player making a single decision. An additional layer of complexity can be created by asking the players to play the game in multiple rounds, with the winner being whoever wins the greatest number of rounds. In this new *iterated prisoner's dilemma*, a strategy is not simply one of {defect, cooperate}, but is any function of the game state that returns a decision. For each round, the players uses his strategy to generate a move.

Let's discuss some of the strategies that may be used when playing the iterated prisoner's dilemma. The two simplest strategies are to always defect or to always cooperate, regardless of any other factors. Another simple strategy, which is also independent of the game state, is to choose randomly between defecting and cooperating, possibly weighting in favor of one choice or the other, such that cooperation is chosen with probability p .

Most sophisticated strategies take advantage of the game state. For instance, another strategy is to cooperate on the first turn, and for each subsequent turn, look at the opponent's history, and do whatever the opponent has done a majority of the time.

In a famous paper, Axelrod (1984) asked game theorists to submit strategies for the iterated prisoner's dilemma in the form of computer programs. After collecting the entries, Axelrod held a round-robin tournament, where each strategy played once against each other strategy (including itself), for a 100 round game.

Axelrod found that the most successful strategy was tit-for-tat, a strategy that cooperates on the first turn, and then on each subsequent turn looks at the opponent's history and chooses whatever move the opponent did on the previous turn. Tit-for-two-tats, a minor variation of tit-for-tat, was also rather successful in the tournament. Tit-for-two-tats is identical to tit-for-tat, except that it defects if and only if the opponent defected on the past two turns.

Axelrod pondered why tit-for-tat was such a successful strategy, and found a few common patterns among successful strategies: they were *nice*, and *forgiving*. A nice strategy is one that is never to be the first to defect. If we look at tit-for-tat, it is never the first to defect because it initially cooperates and then follows the opponent move for move. A forgiving strategy is one that does not hold a grudge: if the opponent defects, it remains open to the possibility of future cooperation. The most extreme case of a begrudging strategy is one that, after an opponent defects once, defects on every turn without ever turning back.

Note that, while tit-for-tat is an all-around robust strategy (performing well across a wide variety of opponents), that the success of any strategy is highly dependent

on the context of the game – specifically, the score of a player depends highly on what the strategies the other players are using. Let’s examine a few simple cases:

When a player who always defects plays against an opponent who plays tit-for-tat, then on the first round the defector defects while tit-for-tat cooperates, favoring the defector. On all subsequent turns, both players defect, result in equal performance. Thus, over the course of a tournament consisting of only tit-for-tat and a player who always defects, the defector will come out ahead.

Similarly, a player who cooperates on every turn will perform terribly when playing against an opponent who always defects, but marvelously against an opponent who also always cooperates.

In Axelrod’s original tournament, each player played once against itself, and once against each of the other opponents. Thus each strategy was a unique member in the population of all strategies. Mimicking nature, we can allow each strategy to have multiple representatives in the population. In the static case, we just choose a distribution of strategies, and watch them play against each other. Since the dynamics of the game are highly dependent on the set of strategies that are playing, we should see interesting results even with this modification. I implemented Axelrod’s round robin tournament, and extended it to allow for arbitrary populations of strategies, all playing round robin. I found, as expected, that the scores were highly dependent on the context, and I confirmed that the noted special cases, as well as a few others, in fact produced the intended results.

Another layer of complexity can be added by making the distribution of strategies within a population dynamic, rather than static as in the previous example. With a dynamic distribution of players, we need some way to choose the composition of the next generation. Drawing from the machine learning literature, I implemented a few different methods:

A simple method for choosing the next generation is to perform a round robin tournament, keep the best 50% of strategies, and allow each of those strategies to generate a second copy of itself to insert into the population. Another variation of this theme is to allow the populations to introduce new members into the population, proportional to their scores in the round robin tournament, effectively boosting the status of high performers. In these ways, successful strategies are rewarded by becoming more prevalent in the next generation (so long as they perform well against the new distribution of strategies). But what if a strategy plays poorly against the initial distribution of strategies, but would play well against later generations? Rosin and Belew (1995) introduced the idea of keeping a hall-of-fame, in which the best member of each generation is made immortal,

playing in every generation. While this is not particularly useful for our limited dynamically-changing population distribution, it becomes very valuable when we add the final layer of complexity to our tournament: evolving strategies.

Axelrod (1987) uses a bitstring to represent strategies for the prisoner's dilemma, and allows these strategies to evolve over many generations. I leveraged the functionality that I had already built into my program to create a tree-based representation of strategies. The foundation on which I built new strategies were (1) mixed strategies, and (2) switched strategies.

A mixed strategy is a meta-strategy, which produces a new strategy given three inputs: (1) the first strategy, (2) the second strategy, and (3) the probability p that, on any given turn, the mixed-strategy should make use of the first strategy. The second strategy is used with probability $1 - p$.

A switch strategy is another meta-strategy, which produces a new strategy given three inputs: (1) the first strategy, (2) the second strategy, and (3) the turn number to switch from only using the first strategy to only using the second strategy.

Given these two methods of generating meta strategies, we can generate arbitrarily complex strategy trees of all possible forms. For instance, we can generate a strategy that cooperates on the first round, defects on the second, and then plays tit-for-tat with probability 0.60, otherwise playing go-by-majority. First we create the mixed strategy `mixed(tit-for-tat, go-by-majority, 0.60)`, and then wrap it in two make-switched strategies for 1 turn each to achieve the static opening moves.

The benefit of using this tree-based method is that only legal strategies are ever generated, saving us the trouble of testing dysfunctional strategies that don't consistently make legal moves.

I created code (see attached) that implements various strategy combination procedures, including mutating strategies (randomly replacing branches of the strategy with new strategies), crossover (interchanging branches from two different strategies), and random replacement/generation of entire strategies (randomly creating a new strategy tree, and replacing a poorly-performing strategy with the new random strategy).

When I ran the simulation, I began with a distribution of "known" strategies (the ones used in the Axelrod tournament). Tit-for-tat, or minor/inconsequential variations on tit-for-tat tended to dominate the tournament. For instance, sometimes late generations would be filled with mixed strategies that combined many levels of tit-for-tat, with the occasional use of go-by-majority. When the trees become large, it becomes hard to give an intuitive explanation of what strategy is being used, other than by showing the tree or just claiming that it mixes many strategies.

In all cases, there was a large stochastic element; the results were never the same twice in a row.

As I write this paper, I realized that I missed one of the most interesting cases, that I would like to explore in the future: creating a first generation full of randomly-created strategies, withholding the complex and robust strategies and then running the simulation. It would be very interesting to see if the system can evolve complex strategies like tit-for-tat, or pavlov, only using very simple primitives like all-defect, poorTrustingFool, and goByMajority.

References

- Axelrod, R. (1984). *The Evolution of Cooperation*.
- Axelrod, R. (1987). The evolution of strategies in the iterated prisoner's dilemma. *Genetic Algorithms and Simulated Annealing, Research Notes in Artificial Intelligence*.
- Pareto, V. (1906). *Manuale di Economia Politica*.
- Poundstone, W. (1993). *Prisoner's Dilemma*.
- Rosin, C. and Belew, R. (1995). New methods for competitive coevolution. *Evolutionary Computation*.